

BACHELORARBEIT

# EVOLUTION VON SPRACHE IN ROBOTERSCHWÄRMEN

---

von **FABIAN KERSTHOLT**

eingereicht am Donnerstag, 29. Juli 2010 beim  
Institut für Angewandte Informatik und Formale Beschreibungsverfahren (AIFB)  
des Karlsruher Instituts für Technologie

Referent: Prof. Dr. Hartmut Schmeck  
Betreuer: Dipl.-Inf. Lukas König

Heimatanschrift:  
Bergmecke 30  
59872 Meschede

Semesteranschrift:  
Josef-Schofer-Str.2  
76187 Karlsruhe



# Inhaltsverzeichnis

<b>1</b>	<b>Abstract</b>	<b>7</b>
<b>2</b>	<b>Einleitung</b>	<b>11</b>
<b>3</b>	<b>Grundlagen</b>	<b>19</b>
3.1	Evolutionäre Algorithmen . . . . .	19
3.1.1	Rekombination . . . . .	21
3.1.2	Mutation . . . . .	23
3.1.3	Selektion . . . . .	24
3.2	Evolutionäre Robotik . . . . .	25
3.3	MARB . . . . .	26
3.4	Die Simulationsumgebung . . . . .	27
<b>4</b>	<b>Analyse und Entwurf</b>	<b>31</b>
4.1	Sensorik . . . . .	31
4.2	„Soziales Szenario“ . . . . .	37
4.3	„Futtersuche Szenario“ . . . . .	38
4.4	Sonstige Implementierungen . . . . .	40
4.4.1	Platzieren der Futterplätze . . . . .	41
4.4.2	Abnahme des Futters . . . . .	43
4.4.3	Visualisierung . . . . .	44
<b>5</b>	<b>Evaluation</b>	<b>47</b>
5.1	Methodik . . . . .	47
5.1.1	„Soziales Szenario“ . . . . .	47
5.1.2	„Futtersuche Szenario“ . . . . .	49
5.1.3	Auswertung & Gütemaß . . . . .	49

5.2	Ergebnisse „Soziales Szenario“ . . . . .	52
5.2.1	„Taub“ Test . . . . .	52
5.2.2	Broadcast Dauer . . . . .	53
5.3	Ergebnisse „Futtersuche Szenario“ . . . . .	55
5.3.1	Verhaltensweise 1: Fahren . . . . .	55
5.3.2	Verhaltensweise 2: Stoppen . . . . .	57
5.3.3	Verhaltensweise 3: Drehen . . . . .	57
5.3.4	Verhaltensweise 4: Stehen . . . . .	61
5.3.5	Zusammenfassung der Ergebnisse . . . . .	62
5.4	Platzieren der Futterplätze . . . . .	63
<b>6</b>	<b>Zusammenfassung</b>	<b>67</b>
6.1	Ausblick . . . . .	67
<b>7</b>	<b>Anhang</b>	<b>71</b>
7.1	Veraltete Verhaltensweisen . . . . .	71
7.1.1	Verhaltensweise: Drehen . . . . .	71
7.2	Standardparametersatz . . . . .	75
	<b>Literaturverzeichnis</b>	<b>77</b>





# 1 Abstract

In dieser Arbeit wird ein dezentraler onboard Evolutionsansatz verwendet, um zu untersuchen, wie sich Sprache in Roboterschwärmen entwickelt. Die hierzu durchgeführten Versuche wurden im FMG<sup>1</sup> Framework von Lukas König ausgeführt, welche auf der Simulation von Jasmine-IIIp [KJKL08] Robotern basiert. Diese werden von sogenannten Moore Automaton for Robot Behavior (vgl. Kap. 3.3) -kurz MARB- gesteuert, welche im Laufe der Evolution schrittweise weiterentwickelt werden, um sich so der gegebenen Problemstellung anzupassen.

Um das Ziel der Sprachentwicklung zu erreichen, werden zuerst mehrere akustische Sensoren, sowie die Möglichkeit einen Broadcast zu senden, implementiert. Die Sensoren erfüllen dabei den Zweck den Broadcast zu empfangen und über die Lautstärke die Entfernung des Emittenten zu bestimmen. Der Broadcast ist von allen Robotern innerhalb der wählbaren Broadcast-Reichweite hörbar und breitet sich gleichförmig in alle Richtungen aus. Da die Roboter von Endlichen Automaten gesteuert werden, wurde ein neuer Zustand hinzugefügt, welcher den Broadcast startet. Damit die Roboter während des Sendens weitere Aktionen durchführen können, bleibt der Broadcast für eine bestimmte Anzahl von Zyklen aktiv, bevor er abgestellt wird. Jeder Aufruf des aktivierenden Zustandes setzt dabei die verbleibenden Sendezyklen wieder auf den Anfangswert.

Diese Realisation wird in zwei verschiedenen Szenarien getestet. Im ersten Fall, dem sogenannten „Sozialen Szenario“, besteht die Aufgabenstellung der Roboter darin, sich von Wänden fernzuhalten, die Nähe anderer Roboter zu suchen und sich gleichzeitig noch zu bewegen. Da die Roboter mit ihren bisherigen Infrarotsensoren nur Abstände und nicht die Art eines Hindernisses feststellen können, muss diese Differenzierung unter Hinzunahme des Broadcasts geschehen. Um die Aufgabe schwieriger zu gestalten, werden zusätzliche Wände in der Umgebung verteilt, die in Form und Größe Robotern ähneln. Das zweite

---

<sup>1</sup>Finite Moore Automaton Generator

Testszenario ist ein „Futtersuche Szenario“. Hierzu wird ein zusätzlicher Futtersensor implementiert, der die Entfernung zu Futterplätzen angibt, welche in der Umgebung verteilt sind. Diese sind in Anzahl und Größe variabel, jedoch immer rund. Aufgabe ist hier, gemeinsam mit anderen Robotern auf einer solchen Futterstelle zu stehen, die allerdings nur ein begrenztes Maß an Futter enthält. Sobald sie eine gewisse Zeit benutzt wurde, verschwindet sie. Zum Ausgleich wird in diesem Fall eine neue Futterstelle zufällig platziert, sodass die Anzahl der aktiven Futterstellen konstant bleibt. Mit ausreichender Sichtweite des Futtersensors ist diese Aufgabe auch ohne den Einsatz von Sprache lösbar, da sich dann das optimale Verhalten der Gruppe lediglich aus dem optimalen Verhalten jedes Individuums zusammensetzt. Jeder Roboter würde zu der Futterquelle fahren, die er gerade sieht. Sobald allerdings der Futtersensor nur eine kurze Reichweite besitzt, sieht ein Großteil der Population keine der Futterquellen, da die Wahrscheinlichkeit eine Futterquelle durch zufälliges Umherfahren zu entdecken sinkt. Nun macht es für die Roboter Sinn, dem Rest der Population die Existenz einer Futterquelle über akustische Signale mitzuteilen. Die fehlende Futtersensorlänge wird sozusagen durch Sprache kompensiert. Voraussetzung ist natürlich, dass der Radius, über den sich Roboter untereinander verständigen können, deutlich höher ist als der des Futtersensors.







## 2 Einleitung

Bis in das späte 18. Jahrhundert herrschte in der Wissenschaft die sogenannte Schöpfertheorie vor, um das Entstehen von Leben auf der Erde zu erklären. So sollten alle Organismen von einem Schöpfer geschaffen worden sein und vor allem konstant bleiben. Fossile Funde wurden beispielsweise als von Naturkatastrophen ausgelöschte Arten interpretiert. Erster Kritiker dieser Theorie war Lamarck, der in seinem Werk „Philosophie Zoologique“ [Lam09] diese Artenkonstanz anzweifelte. Er ging davon aus, dass auf der jungen Erde erste, einfache Organismen durch Urzeugung entstanden, aus denen dann durch aktive Anpassung an die Umwelt (durch Gebrauch und Nichtgebrauch eines Organs), die heute lebenden Arten stufenweise durch Umwandlung entstanden seien. Doch weder Lamarck noch Darwin, welcher gut 50 Jahre später mit seinem Werk „On the Origin of Species“ [Dar59] das Prinzip der natürlichen Selektion einführte, konnten zunächst einen echten Wandel der vorherrschenden Meinung auslösen. Trotzdem wird das Werk Darwins als Grundstein der **Evolutionstheorie** angesehen, obwohl er in der ursprünglichen Fassung das Wort Evolution nicht verwendete. Etwa zeitgleich (1866) stellte Mendel durch Beobachtungen bei der Kreuzung von Gartenerbsen die sogenannten „Mendel’schen Regeln“ auf und begründete damit die Anfänge der Genetik. Anfangs hatte die Genetik keinen Einfluss auf Darwins Theorie, vor allem weil die Veröffentlichung der „Mendel’schen Regeln“ völlig unbeachtet blieb. Erst deren Wiederentdeckung und das 1901 erstmals beobachtete Phänomen der Mutation rückten die Genetik in den Mittelpunkt. In der Mitte des 20. Jahrhunderts entdeckten James D. Watson und Francis H.C. Crick die DNA sowie den genetischen Code. Die Erkenntnisse der Genetik wurden mit der Selektionstheorie Darwins in Einklang gebracht. Es entstand die Synthetische Theorie der Evolution. Zufallsereignisse wie Rekombination und Mutation liefern das Ausgangsmaterial für die natürliche Selektion. Unter Mutation versteht man eine spontane, ungerichtete Veränderung des genetischen Materials (vgl. Kapitel 3.1.2). Rekombination bedeutet die Neukombination des Erbgutes bei der sexuellen Fortpflanzung (vgl. Kapitel 3.1.1). Beide

erhöhen die Diversität, also die Vielfalt des Erbgutes. Ihr wirkt die Selektion entgegen.

Das Erscheinungsbild eines Individuums wird als Phänotyp bezeichnet und wird von zwei Komponenten festgelegt. Einerseits beeinflusst die Umwelt den Organismus. Zum Beispiel hat seine Lebensweise Auswirkungen auf körperliche Merkmale wie beispielsweise Gewicht oder Größe. Dies wird Modifikation genannt. Sie sind nicht erblich, da sie keine Auswirkungen auf die Gene haben. Einen weitaus größeren Einfluss auf den Phänotyp hat jedoch der Genotyp. Dieser beschreibt das Erbbild eines Organismus und repräsentiert seine genetische Ausstattung, also die Kombination aller Allele. Ein Allel ist dabei die Zustandsform oder Ausprägung eines Gens. Die Menge aller Allele einer Population bilden den Genpool. Als Population wird eine Gruppe von Individuen bezeichnet, welche eine Fortpflanzungsgemeinschaft bilden, also zur gleichen Art gehören und in einem abgegrenzten Raum leben.

Das Prinzip der natürlichen Selektion besagt, dass sich in einer Population diejenigen Individuen häufiger vermehren, die besser an die Umweltbedingungen angepasst sind. Somit gehen ihre Gene, welche die Informationen für die bessere Anpassung enthalten, häufiger in den Genpool der folgenden Generationen ein, sodass auch die Nachkommen vermehrt diese positiven Eigenschaften tragen. Auf diese Weise wird eine Population im Laufe der Generationen immer besser an die Umweltbedingungen angepasst.

Die biologischen Grundbegriffe und Theorien werden von der Evolutionären Algorithmik (vgl. Kapitel 3.1) verwendet. Diese findet Anwendung in der Evolutionäre Robotik (vgl. Kapitel 3.2) um Kontrollprogramme für autonome Roboter zu erzeugen. Da die Anforderungen, die an Roboter gestellt werden, immer weiter an Komplexität zunehmen, wird es zusehends schwieriger entsprechende Programme manuell zu entwerfen. Zum einen liegt dies an den immer schwieriger werdenden Aufgaben, die es zu lösen gilt. Zum anderen ist die Umgebung mit der die Roboter interagieren häufig nicht vollständig bekannt oder ständigem Wandel unterzogen. Sobald nicht nur ein Roboter, sondern ein Schwarm eine bestimmte Problemstellung lösen soll, kommt erschwerend hinzu, dass das aus Interaktionen vieler unabhängig agierender Individuen emergierende Verhalten des gesamten Schwarmes kaum vorhersehbar ist. Aus diesen Problemen leitet sich die **Motivation** ab, Robotern die Möglichkeit zu geben, sich eigenständig an die gegebene Aufgabenstellung anzupassen, also sich im Sinne einer Evolution weiterzuentwickeln.

Evolutiv unterscheiden sich die Primaten von anderen Lebewesen durch die Fähigkeit ih-

ren Daumen den anderen Fingern gegenüberzustellen. Der Mensch wiederum differenziert sich von anderen Primaten durch seine kognitiven Fähigkeiten. Wie der Sprachwissenschaftler Benjamin Lee Whorf bereits in den 1930er Jahren postulierte, seien bestimmte Denkkonzepte demnach überhaupt nicht zugänglich, wenn die Sprache dafür keinen Ausdruck kennt [Kie96, S. 481]. Doch nicht nur der Mensch hat komplexe Kommunikation entwickelt. Auch im Tierreich sind komplexe Verhaltensweisen entstanden, welche allerdings nicht zwingend auf akustischen Signalen basieren. Bienen teilen ihren Artgenossen beispielsweise durch einen komplexen Tanz die genaue Lage, Art und Qualität einer gefundenen Futterquelle mit [LK58, S. 405]. Ameisen markieren den Weg zu einer Futterstelle durch Pheromone oder taktile Kommunikation ihrer Fühler. Erdmännchen haben für verschiedene Bedrohungen unterschiedliche Pfeifsignale entwickelt, um andere Tiere der Herde zu warnen und Wale können selbst noch über mehrere hundert Kilometer durch Laute miteinander kommunizieren. In vieler Hinsicht ist Sprache oder Kommunikation im Allgemeinen Voraussetzung für komplexe Verhaltensweisen in Gruppen von Individuen. Wenn die Sprache in der Natur eine so große Bedeutung hat, kann sie potenziell auch zum Lösen von Problemstellungen in Roboterschwärmen benutzt werden. Aus diesen Gründen ist das Thema dieser Arbeit die Evolution von Sprache in Roboterschwärmen.

Der Begriff der **Kommunikation** wird je nach Wissenschaftsdisziplin auf unterschiedliche Weise definiert, je nachdem welche der unterschiedlichen Aspekte der Kommunikation im Vordergrund stehen. In der Nachrichtentechnik beschränkt sich die Definition von Kommunikation beispielsweise auf den reinen Austausch von Nachrichten, die mit Hilfe von Sende- und Empfangsgeräten übermittelt werden können [Wea49, S. 3]. Dabei steht die syntaktische Ebene eines Kommunikationsvorgangs im Vordergrund. In der Psychologie hingegen wird eher die Metaebene der Kommunikation, also die „persönliche Bedürfnisbefriedigung der Kommunizierenden“ [Sch06, S. 48] betrachtet und weniger die reine Vermittlung reiner Sachinhalte. In dieser Arbeit sei Kommunikation als reiner Austausch von Information zwischen zwei Individuen, im Speziellen zweier Jasmine-IIIp Roboter definiert. **Sprache** hingegen ist die „symbolische Repräsentation der Welt in unseren Köpfen“ [BB06, S. 25]. Im Prinzip referenzieren wir mit Sprache eine konkrete Situation der Umwelt und können so die gewonnenen Erfahrungen samt ihrer Konsequenzen mitteilen, ohne dass die Erfahrungen von neuem gemacht werden müssen [BB06, S. 25]. Sprache begrenzt sich hierbei nicht nur auf eine Abfolge akustischer Signale, welche einer komplexen Zuordnungsvorschrift unterworfen ist. Vielmehr existiert auch nonverbale Sprache,

beispielsweise die rein aus Gestik und Mimik bestehende Gebärdensprache, welche natürlich auch fest definierten Regeln folgt. Mittlerweile gehen Wissenschaftler davon aus, dass auch die menschliche Sprache einen nonverbalen Ursprung hat [Arm08]. So haben Experimente mit Schimpansen gezeigt, dass sie wesentlich besser gestikulieren als sprechen [Bal10].

In dieser Arbeit soll also untersucht werden, ob Roboter in der Lage sind zu sprechen, ob sie also über den reinen Austausch von Signalen, im Sinne von Kommunikation, auch eine Verknüpfung zwischen diesen Signalen und ihrer Bedeutung innerhalb der Umgebung herstellen können. Dabei liegt der Schwerpunkt auf rein akustischen Signalen und somit verbaler Sprache.

In den letzten Jahren hat die Evolutionäre Schwarmrobotik immer mehr an Bedeutung gewonnen. Maßgeblich dafür verantwortlich ist das sogenannte „Design Problem“ [Tri08, Kap. 4.1]. Dieses beschreibt die Schwierigkeiten, die sich beim Entwurf von Kontrollmechanismen für Schwarmroboter ergeben. Zum einen muss das gewünschte Verhalten des Schwarms als Resultat der Interaktion der einzelnen Individuen ausgedrückt werden. Zum anderen muss das so ermittelte, gewünschte Verhalten der einzelnen Roboter entworfen werden. Die Schwierigkeit ist hierbei nicht nur die Interaktion der einzelnen Roboter untereinander, sondern auch ihre Interaktion mit ihrer Umgebung.

Die Schwarmrobotik ist ein Lösungsansatz dieses „Design Problems“ und umgeht die Problematik das gewünschte Verhalten in Einzelverhalten der Individuen transformieren zu müssen. Diese Aufgabe wird vom evolutionären Prozess übernommen, welcher nur diejenigen Verhaltensweisen fördert, welche im Zusammenspiel miteinander und der Umgebung zum gewünschten globalen Verhalten führen [Tri08, Kap. 4.1].

Die Vorteile der Evolutionären Robotik wurden in der Vergangenheit in vielen Szenarien zur Anwendung gebracht. Dabei wird in der Literatur grundlegend zwischen Versuchen unterschieden, welche in einer Simulationsumgebung ablaufen, und denen, die auf realen Robotern durchgeführt werden. Simulation haben den Vorteil, dass sie kostengünstiger, eventuell schneller und wesentlich variabler sind. Außerdem, so Walker u.a. [WGW03, S. 3], erlauben in Simulationen durchgeführte Versuche dem Forscher sich auf das Entwickeln von Kontrollmechanismen zu konzentrieren, da sie sich nicht mit physischen Problemen auseinandersetzen müssen, welche bei Versuchen mit realen Robotern aufkommen könnten. Aus diesen Gründen wurden die in dieser Arbeit vorgestellten Versuche ausschließlich

in einer Simulationsumgebung durchgeführt, obwohl sie theoretisch auf realen Robotern verwirklicht werden könnten.

Weiterhin unterscheiden Hoen u.a. [HTP<sup>+</sup>05] in ihrer Arbeit zwischen kooperativen und kompetitiven Multiagentensystemen. In kooperativen Systemen erhalten die Individuen dieselbe Aufgabe, wo hingegen in kompetitiven Systemen konträre Aufgabenstellungen zu lösen sind. Wie in den Kapiteln 4.2 und 4.3 näher beschrieben wird, sind die in dieser Arbeit zu lösenden Problemstellungen kooperativer Natur. Waibel u.a. [WGW03, S. 649] unterteilen die kooperativen Aufgabenstellungen weiter nach ihren Kosten für das kooperierende Individuum, nämlich in Aufgabenstellungen ohne verbundene Kosten und altruistische Aufgabenstellungen, bei denen die Kosten gegenüber dem Nutzen der Kooperation überwiegen. Die in dieser Arbeit vorgestellten Aufgaben beinhalten sowohl Kosten als auch Nutzen für Kooperation. Wie in Kapitel 4.1 vorgestellt, büßen die Individuen für Kommunikation einen Teil ihrer Beweglichkeit ein, allerdings sind die Kosten hierfür schwer quantifizierbar. Es wurde allerdings versucht, die Kosten für Kooperation geringer als den resultierenden Gewinn zu halten.

Weiterhin untersuchen Waibel u.a. wie verschiedene Selektionsmechanismen die Performanz eines Multiagentensystems beeinflussen. Hierzu führten sie Versuche durch, welche unterschiedliche Level an Kooperation erforderten und kamen unter anderem zu dem Schluss, dass mit steigendem Aufwand für Kooperation homogene Populationen bessere Ergebnisse liefern als heterogene Gruppen. Homogene Populationen seien somit, so Waibel u.a. [WKF09, S. 657], die sichere Wahl, wenn kooperierende Verhaltensweisen evolviert werden sollen. Deshalb wird die Parametrisierung der Versuche dieser Arbeit so gewählt, dass sich die Individuen der resultierenden Roboterpopulation möglichst ähnlich sind.

Im Folgenden werden exemplarisch einige **verwandte Untersuchungen** vorgestellt. Trianni führt in seinem Werk „Evolutionary Swarm Robotics“ [Tri08, Kap. 7.3] das sogenannte „Hole Avoidance“ Problem ein. Bei diesem werden sogenannte „s-Roboter“ [Tri08, Kap. 5] zu einem „swarm-Roboter“ physisch verbunden. Ihre Aufgabe ist es, durch eine mit Löchern versehene Umgebung zu steuern, ohne in diese hineinzufallen. Dabei wird jeder einzelne Roboter von einem neuronalen Netz gesteuert. Die individuellen Sensorreichweiten, mit denen ein Loch erkannt werden kann, sind dabei so gering, dass ein Roboter ein Loch erst erkennen würde, wenn er in dieses hinein fiel. Um dies zu kompensieren, besitzen die Roboter die Möglichkeit zu kommunizieren. Die Ergebnisse zeigen, dass die so entwickelte Sprache sogar performanter als eine manuell erstellte Referenz ist.

Floreano u.a. [FMMK07] verwenden visuelle Kommunikation via blauen LEDs in einer Umgebung mit zwei verschiedenen Futterquellen. Eine dieser Futterquellen enthält „Nährstoffe“ und hat somit einen positiven Effekt auf die Güte der Roboter, eine andere „Gift“ und somit den gegenteiligen Effekt. In diesem Szenario sind zwei verschiedene Verhaltensweisen zu beobachten. Neben der klassischen, selbstlosen Kommunikation, welche den anderen Individuen die Position der Futterquelle bzw. der Giftquelle signalisierte, entwickelte sich ebenfalls eine „austricksende“ Verhaltensweise. In dieser, so Floreano, verhalten sich Individuen genau entgegengesetzt. So deaktivieren sie ihre LEDs, wenn sie sich an einer Futterquelle befinden und aktivieren sie, wenn sie sich im freien Raum bewegen. Da die Roboter in den meisten Fällen auf blaue LEDs zusteuern, locken diese „austricksenden“ Roboter andere Individuen auf diese Weise von der Futterstelle, um sich dadurch selbst einen Platz an dieser zu sichern.

Die in dieser Arbeit verwendeten Kontrollmechanismen basieren auf dem in Kapitel 3.3 vorgestellten MARB-Modell. Wie Lukas König u.a. [KJKL08, KS08] in ihren bisherigen Untersuchungen gezeigt haben, ist dieses Modell in der Lage robustes Verhalten auf Schwarmrobotern zu erzeugen. So wurde beispielsweise erfolgreich ein „Collision-Avoidance“ sowie ein „Gatepassing“ Verhalten entwickelt. Hierbei war die Aufgabe, ein zentrales Tor in der Umgebung zu durchfahren, ohne mit anderen Individuen zu kollidieren. In weiteren Studien [WK10] wurde bereits gezeigt, dass Sprache durch das MARB-Modell realisiert werden kann, sodass Roboter von Wänden unterschieden werden können. Diese Tests sollen nun weiter vertieft werden.

Die **Gliederung** dieser Arbeit ist wie folgt: In Kapitel 3 werden die theoretischen Grundlagen der Arbeit erläutert, bevor in Kapitel 4 auf die implementierten Erweiterungen des bestehenden Frameworks, sowie die den Robotern gestellten Aufgaben, eingegangen wird. In Kapitel 5 werden die durchgeführten Simulationen ausgewertet und die hierzu angewandten Verfahren erläutert. Schließlich werden die Ergebnisse dieser Arbeit in Kapitel 6 zusammengefasst und ein kurzer Ausblick gegeben. Im Anhang, dem 7. Kapitel, wird zum einen der Einfluss der Sensorwertberechnung auf die Evolution von Verhalten an einem konkreten Beispiel illustriert, zum anderen der Standardparametersatz der Simulationen erläutert.







## 3 Grundlagen

In diesem Kapitel wird in die theoretischen Grundlagen dieser Arbeit eingeführt. Die Steuerprogramme der Roboter, welche im FMG Framework simuliert werden, basieren auf Endlichen Automaten und werden durch das MARB-Modell [KS08] beschrieben. Diese werden verwendet, um Verhalten auf Robotern online und auf dezentrale Weise zu evolvieren. Deshalb ist dieses Grundlagenkapitel wie folgt strukturiert: Zuerst werden Evolutionäre Algorithmen im Allgemeinen beschrieben. Anschließend wird genauer auf das Teilgebiet der Evolutionären Robotik eingegangen, bevor das MARB-Modell definiert wird. Zuletzt wird die konkrete Umsetzung der vorgestellten Evolutionären Operatoren sowie des MARB-Modells in der genutzten FMG Simulationsumgebung vorgestellt.

### 3.1 Evolutionäre Algorithmen

Evolutionäre Algorithmen nutzen Prinzipien der natürlichen Evolution, um so den optimalen Lösungswert fast beliebiger Optimierungsprobleme zu approximieren. Hierbei sind im Laufe der vergangenen Jahrzehnte mehrere Modelle entstanden, welche sich „alle Begriffe aus der Biologie entlehnen, um [...] Verfahren zur Lösung von Optimierungsproblemen zu beschreiben“ [Wei07, S. 1]. Ein Evolutionärer Algorithmus operiert allgemein auf einer Population. Eine Population ist hierbei analog zur biologischen Definition die Gesamtheit aller Individuen eines umgrenzten Raumes, auch Fortpflanzungsgemeinschaft genannt. Jedes Individuum besteht dabei aus künstlichen Chromosomen. Die Gesamtheit aller Chromosomen eines Individuums wird als Genotyp bezeichnet. Der Genotyp kodiert das Erscheinungsbild eines Individuums, also dessen Phänotyp. Dieser beschreibt eine mögliche Lösung des zu optimierenden Problems. Beispielsweise könnte ein Genotyp die Binärcodierung eines zu optimierenden Funktionswertes, des Phänotypen, darstellen oder wie in Kapitel 3.3 genauer erklärt, aus einem Endlichen Automat bestehen, welcher

eine bestimmte Verhaltensweise erzeugt. Ein Evolutionärer Algorithmus entwickelt die Population iterativ weiter, indem höher bewertete Individuen häufiger reproduziert werden als niedriger bewertete. Um entscheiden zu können, ob eines der Individuen besser als ein anderes ist, wird eine sog. Gütefunktion benötigt. Diese definiert die zu optimierende Problemstellung, charakterisiert also beispielsweise die von Schwarmrobotern zu lösende Aufgabe und muss für jeden möglichen Wert des Suchraumes eindeutig definiert sein. Damit der Optimierungsalgorithmus eine möglichst gute Lösung findet und nicht etwa nur in einem lokalen Optimum sucht, ist die Vielfalt einer Population von entscheidender Bedeutung. Dabei reicht es nicht, dass mehrere Individuen existieren, sondern sie müssen möglichst über den gesamten Suchraum verteilt sein. Ein Maß dafür, wie gut die Verteilung einer Population im Suchraum ist, ist die *Diversität*. Generell existieren verschiedene Definitionen, beispielsweise wird der mittlere Abstand aller Individuen der Population [Wei07, S. 62], manchmal auch der absolute Abstand, gebildet. Der Abstand zweier Individuen beträgt dabei meist die Summe der Differenzen der einzelnen Chromosomen. Im konkreten Fall des MARB-Modells ist die Anzahl verschiedener Endlicher Automaten innerhalb der Population ein Maß für die Diversität. Um Diversität innerhalb der Population zu schaffen, werden die Individuen zufällig verändert. Die Stärke der Veränderungen können sowohl antiproportional zur Fitness eines Individuums sein oder es werden, wie beispielsweise im MARB-Modell, alle Individuen mit gleicher Wahrscheinlichkeit verändert. Für diese Veränderungen werden zwei weitere generische Operatoren genutzt: Rekombination und Mutation. Da beide Operatoren Zufallszahlen nutzen, um neue Individuen zu erzeugen, sind Evolutionäre Algorithmen nicht deterministisch und bieten als stochastische Lösungsverfahren deshalb insbesondere keine garantierte Lösungsqualität. Ihr Vorteil liegt aber darin, dass an das zu optimierende Problem relativ wenige formale Restriktionen gestellt werden. Der allgemeine Grundablauf eines Evolutionären Algorithmus funktioniert wie folgt:

Zu Beginn der Optimierung muss eine Startpopulation generiert werden. Dies kann rein zufällig geschehen. Sobald allerdings zusätzliche Informationen über den Suchraum zur Verfügung stehen, ist es meist sinnvoll diese auch auszunutzen. So ist es allgemein nicht sinnvoll, Individuen an den Randpunkten des Suchraums zu platzieren, da diese von Zufallsgeneratoren meist nicht komplett abgedeckt werden. Anschließend muss die Startpopulation nach der zuvor festgelegten Fitnessfunktion bewertet werden. Nun startet der iterative Prozess. Zuerst werden einige Individuen selektiert (vgl. 3.1.3), bevor diese rekombiniert (vgl. 3.1.1) sowie mit einer gewissen Wahrscheinlichkeit mutiert (vgl. 3.1.2)

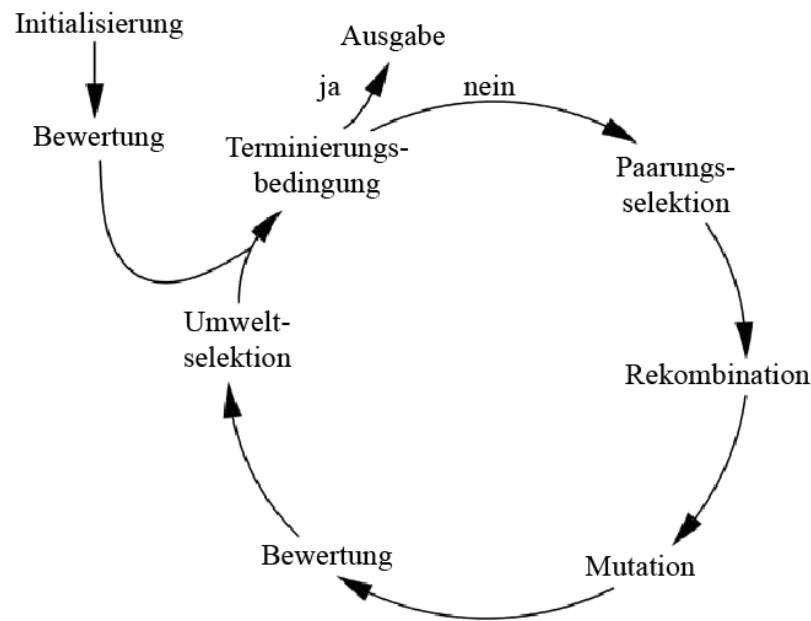


Abbildung 3.1: Schematische Darstellung des Zyklus bei Evolutionären Algorithmen [Wei07, Seite 25]

werden, um neue Individuen zu generieren. Diese Mutanten werden wiederum bewertet, bevor die gesamte Population der Umweltselektion unterzogen wird. Am Ende dieses Schrittes stehen die besten  $n$  Individuen der Kinderpopulation oder der Kinder- und Elternpopulationen, je nach Selektionsmechanismus. Diese bilden die neuen Eltern für den nächsten Iterationsschritt, welcher gestartet wird, falls das Abbruchkriterium nicht erreicht ist. Solche Kriterien sind typischerweise Fitnesswertverbesserungen, also Qualität des Verhaltens, oder Iterationsschritte, also Zeit.

### 3.1.1 Rekombination

Ziel der Rekombination ist es neue, potenziell bessere Lösungen zu generieren. Hierzu wird durch Mischung des genetischen Materials von mindestens zwei Individuen der aktuellen Population, den sog. Eltern, mindestens ein neues Individuum erzeugt, auch Kinderindividuum genannt. Je nach Datenart des Genotyps der Phänotypen können unterschiedliche Arten der Rekombination angewandt werden. Dabei kann es durchaus vorkommen, dass eine sinnvolle Rekombination der Genotypen sehr komplex und schlecht zu automatisie-

ren ist. Dies ist beispielsweise auch in der verwendeten Simulationsumgebung der Fall, weshalb dort zum Generieren neuer Individuen lediglich ein Mutationsoperator verwandt wird, welcher in Kapitel 3.4 genauer erläutert wird. Im Folgenden werden exemplarisch drei verschiedene Gruppen an Rekombinationsoperatoren vorgestellt, um den Prozess der Rekombination näher zu erläutern.

Die *kombinierende Rekombinationsoperatoren* ähneln am ehesten der aus der Natur bekannten Rekombination. Sie setzt das neue Individuum direkt aus verschiedenen Eigenschaften, sei es auf Basis des Phänotyps oder Genotyps, zusammen. Idealerweise erbt so das Kind nur die besten Eigenschaften seiner Eltern und nähert sich so immer weiter dem Optimum an. Das Maß an Diversitätssteigerung dieses Operators hängt in starkem Maße von der bereits in der Population vorhandenen Diversität ab, da durch die reine Kombination von Ausprägungsmerkmalen keine neuen Bereiche des Suchraumes erreicht werden können [Wei07, S.80].

Im Gegensatz dazu generieren *interpolierende Operatoren* Kinderindividuen, indem sie die Charakteristika der Eltern so kombinieren, dass die so entstehenden Eigenschaften zwischen denen der Eltern liegen. Sei beispielsweise der zweidimensionale Suchraum aller reellen Zahlen im Intervall  $[0,2]$  gegeben, bei denen ein Elter den Phänotyp  $(0,0)$ , der andere  $(1,1)$  trägt. Weiterhin operieren alle Rekombinationsoperatoren direkt auf dem Phänotyp. Ein kombinierender Rekombinationsoperator würde nur Kinderindividuen generieren, deren Phänotyp in den Ecken des Teiles des Suchraumes liegen, welcher durch die Elternindividuen aufgespannt wird, also entweder die Werte  $(0,0)$ ,  $(0,1)$ ,  $(1,0)$  oder  $(1,1)$  annimmt. Die Kinder, welche durch einen interpolierenden Operator erzeugt wurden, würden hingegen in der Menge  $\{(x,y) \in [0,1]^2 \mid x = y\}$  also auf der gesamten Strecke zwischen den beiden Elternindividuen liegen. An diesem Beispiel wird deutlich, warum, so Weicker [Wei07, S. 81], die kombinierenden Operatoren Diversität erhalten, interpolierende Operatoren hingegen eher dazu führen, dass die Population gegen ihren Schwerpunkt konvergiert. Doch auch die interpolierenden Operatoren haben durchaus ihre Daseinsberechtigung, da so beispielsweise durch Mutationen (vgl. 3.1.2) erzeugte große Sprünge im Suchraum wieder abgeschwächt werden können.

Die dritte Art der Rekombinationsoperatoren, die sogenannten *explorativen Operatoren*, benötigen, im Gegensatz zu den beiden bisher vorgestellten, Grundannahmen über den vorliegenden Suchraum. Daraus versuchen sie abzuleiten, wo im Suchraum Güteverbesserungen zu erwarten sind. Auf diese Weise können die Kinderindividuen auch Sprünge

in bisher noch unbekannte Teile des Suchraumes vollziehen und so die Diversität der Population erhöhen. Theoretisch kann es hierbei auch auftreten, dass die Kinderindividuen außerhalb des Suchraumes liegen, weshalb die neuen Individuen im Allgemeinen auf Zulässigkeit überprüft werden, um eventuell den Rekombinationsschritt zu wiederholen oder durch alternative Mechanismen die Zulässigkeit wiederherzustellen. Laut Weicker [Wei07, S. 83] haben die explorativen Operatoren deshalb den Nachteil, dass Individuen immer weiter an den Rand rücken und so die Performanz negativ beeinflussen.

### 3.1.2 Mutation

Wie im vorherigen Kapitel 3.1.1 gezeigt, ist es mittels Rekombination nur in seltenen Fällen, nämlich nur wenn zusätzliche Informationen über den Suchraum zur Verfügung stehen, möglich neue Bereiche des Suchraumes zu erreichen [Wei07, Kap. 3.1.6]. Aus diesem Grund wird ein weiterer evolutionärer Operator benötigt: die Mutation. Diese verändert die bisher gefundenen Lösungen zufällig, um so Sprünge im Suchraum zu ermöglichen. Eine Mutation ist allgemein eine Abbildung des genotypischen Suchraumes in sich selbst. Die genaue Funktion eines Mutationsoperators hängt stark von der gewählten Repräsentation des Phänotyps durch den Genotyp ab. Meist wird eine gleichverteilte Zufallszahl  $Z_i \in [0, 1]$  gezogen. Falls  $Z_i \leq p_M$  gilt, wird das gerade ausgewählte Genom  $i$  mutiert, wobei  $p_M$  der zuvor festzulegende Schwellenwert der genbezogenen Mutationswahrscheinlichkeit ist. Typischerweise wird  $p_M$  recht klein gewählt, damit die Population in einem Mutationsschritt nicht zu große Sprünge im Suchraum vollbringt, da dies negativen Einfluss auf die Konvergenz des Algorithmus haben kann [Wei07, Kap.4]. Im einfachsten Fall, falls das Genom ein Binärstring ist, könnte eine Mutation durch die Invertierung des jeweiligen Bits realisiert werden, im reelwertigen Fall beispielsweise durch die Addition einer normalverteilten Zufallsvariablen. Hier wird deutlich, dass die geeignete Wahl des Operators stark von der gewählten Repräsentation der Problemstellung abhängt. Neben der Aufgabe der Suchraumexploration, hat die Mutation die weitere Aufgabe durch kleine Veränderungen die direkte Nachbarschaft des ausgewählten Individuums weiter zu untersuchen und so zu einer Konvergenz des Algorithmus zu führen. Je nachdem wie die Parametrisierung gewählt wird, kann zwischen den beiden Aufgaben unterschieden werden. So führt ein höherer Schwellenwert  $p_M$  tendenziell zu größeren Sprüngen im Suchraum.

Allerdings besteht bisher durchaus noch Uneinigkeit, in welchen Anwendungsgebieten

stärker mutiert und ich welchen stärker rekombiniert werden sollte[BL04, S. 156]. Dementsprechend gibt es auch unterschiedliche Meinungen, in welcher Größenordnung die optimale Parametrisierung eines Mutationsoperators liegen sollte. Wie bereits eingangs erwähnt, gilt es in der Evolutionären Robotik als schwierig, sinnvoll zu rekombinieren, weshalb in der in Kapitel 3.4 vorgestellten Simulationsumgebung lediglich ein Mutationsoperator verwendet wird.

### 3.1.3 Selektion

Der dritte generische Operator, die Selektion trägt nicht zur Diversitätssteigerung bei. Ziel der Selektion ist vielmehr, Individuen zur Paarung oder Mutation auszuwählen und somit den Evolutionsprozess zu möglichst guten Lösungen zu steuern. Generell lässt sich hierbei zwischen zwei verschiedenen Operatoren unterscheiden: stochastischen und deterministischen Selektionsoperatoren.

*Stochastische Selektionsoperatoren* wählen Individuen meist anhand der Fitnesswerte und Zufallszahlen aus. Die beiden gängigsten Operatoren sind dabei die Turniers Selektion und die Roulette-Rad-Selektion. Erstere wählt zufällig zwei Individuen aus, welche anschließend gegeneinander „antreten“. So wird das Individuum mit der höheren Fitness für die weiteren Evolutionsschritte ausgewählt. Haben beide Individuen die gleiche Fitness, entscheidet erneut der Zufall. Bei diesem Selektionsmechanismus werden offensichtlich Individuen mit höherer Fitness bevorzugt. Allerdings ist die Selektionsintensität geringer als bei der Roulette-Rad-Selektion. Dies hat eine langsamere Konvergenz zur Folge und so wird das Risiko verringert in lokalen Optima zu landen. Die Roulette-Rad-Selektion hingegen weist eine wesentlich höhere Selektionsintensität auf. Zuerst wird die totale Fitness  $F$  der Population nach der Formel  $F = \sum_{i=0}^n f(x_i)$  berechnet, wobei  $n$  die Anzahl aller Individuen und  $f(x_i)$  die Fitness des Individuums  $i$  ist. Anschließend wird eine ganzzahlige Zufallszahl  $Z$  aus dem Intervall  $[1, F]$  gezogen. Das Individuum  $k$ , welches die Bedingung  $F(k-1) < Z \leq F(k)$  erfüllt, wird selektiert. Offensichtlich werden bei diesem Selektionsoperator Individuen mit überdurchschnittlich hoher Fitness stark bevorzugt, was speziell in frühen Zyklen der Optimierung schnell zur Konvergenz gegen ein lokales Optimum führen kann. Weiterhin ist die Roulette-Rad-Selektion nur auf positive Fitnesswerte direkt anwendbar.



Im Gegensatz zu den bisher vorgestellten Verfahren wählen *deterministische Selektionsoperatoren* Individuen nur nach Ihren Fitnesswerten aus, indem beispielsweise die  $m \leq n$  besten Individuen der Population gewählt werden. Dies wird meist durch eine deterministische Selektion realisiert.

## 3.2 Evolutionäre Robotik

Die Evolutionäre Robotik betrachtet Roboter als künstliche Organismen, welche ihr Verhalten im engen Zusammenspiel mit der Umgebung entwickeln, insbesondere ohne direkte Einwirkung des Menschen. Der Mensch dient lediglich als Kontrollorgan. Die Hauptschwierigkeit beim Entwerfen von konventionellen, also nicht evolutionären Kontrollautomatismen für autonome Roboter ist die Umgebung [NF00, Kapitel 1.1]. Roboter und Umgebung bilden ein dynamisches System<sup>1</sup>, da die Sensorwerte und damit das weitere Verhalten sowohl von der Umgebung als auch von den vorhergegangenen Aktionen des Roboters abhängen. Deshalb kann das resultierende Verhalten nicht direkt aus den Verhaltensregeln abgeleitet werden. Im Umkehrschluss können für ein gewünschtes Verhalten nicht einfach die entsprechenden Regeln entworfen werden. Um diese Schwierigkeiten zu umgehen, werden einige Grundprinzipien der Evolutionstheorie nach Darwin (vgl. Kapitel 2), welche sich schon in der Natur zum Lösen komplexer Problemstellungen bewiesen haben, genutzt, um Verhalten zu erzeugen.

Gegenüber anderen Formen des maschinellen Lernens, wie beispielsweise „learning classifier“ Systemen, bieten die hier genutzten Evolutionären Algorithmen den Vorteil, auch ohne Testdatensätze arbeiten zu können, die typischerweise in Anwendungsbereichen der Evolutionären Robotik nicht zur Verfügung stehen. Somit wird vom Programmierer kaum eigentliche Programmierarbeit gefordert, sondern lediglich eine Bewertungs- oder Fitnessfunktion, welche auf das gewünschte Verhalten zugeschnitten ist.

---

<sup>1</sup>Wie in [NF00, Kapitel 1.1] definiert

### 3.3 MARB

In diesem Teilgebiet der Evolutionären Robotik werden Endliche Moore Automaten als Kontrollinstanz des Verhaltens der Roboter benutzt. Ein Moore Automat ist allgemein definiert als 7 Tupel  $\mathcal{A} = (Q, \Sigma, \Omega, \delta, \lambda, q_0)$  mit

$Q$	: Endliche Menge von Zuständen	$\Sigma$	: Eingabealphabet
$\Omega$	: Ausgabealphabet	$\delta$	: Ausgabefunktion
$\lambda$	: Ausgabe	$q_0$	: Startzustand

Weiterhin muss  $|Q| < \infty$  sowie  $q_0 \in Q$  gelten [Moo56].

Im Gegensatz zu Mealy Automaten hängt die Ausgabe eines Moore Automaten ausschließlich von seinem Zustand und nicht zusätzlich von der aktuellen Eingabe ab.

Ein *Moore Automaton for Robot Behavior* besteht aus Zuständen und Zustandsübergängen. Ein Zustand besteht aus einer Identifikationsnummer sowie Anweisung. Im konkreten Fall des FMG Frameworks ist jede Anweisung  $q \in Q$  eine „atomare Aktion“, beispielsweise „move, stop, turn left“ [KJKL08, S.88] oder auch „send“. Zu jeder Anweisung wird also eine Ausgabe produziert, welche eine einfache Aktionen wie „Drehen“, aber auch ganze C-Programme beinhalten kann, welche dann von den Robotern ausgeführt werden. Zustandsübergänge bestehen aus einer Bedingung sowie dem Folgezustand und bestimmen ausgehend vom aktuellen Zustand, welcher der nächste Zustand ist. Eine Bedingung ist dabei definiert als:

$$c ::= true | false | z_1 \triangleleft z_2 | (c_1 \circ c_2)$$

$$z_1, z_2 \in B^+ = \{1, \dots, 255\} \cup H \in \{h_1, \dots, h_n\}$$

mit den Sensoren  $H$  und ihren Sensorwerten  $B$

$$\triangleleft \in \{<, >, \leq, \geq, =, \neq, \approx, \not\approx\}$$

wobei  $\approx$  eine Abweichung von  $\pm 5$  bedeutet

$$\circ \in \{AND, OR\}$$

$c_1, c_2$  seien wiederum Bedingungen

Falls keine der Bedingungen der vom aktuellen Zustand abgehenden Zustandsübergängen wahr ist oder keine Zustandsübergänge existieren, ist der Folgezustand  $q_0$  [KJKL08, S. 87].

Wenn diese Endlichen Automaten als Steuerprogramme verwendet werden, interpretiert man einen Automaten als Genom des Roboters, den er kontrolliert, wobei jeder Roboter genau einen Automaten besitzt. Analog sind die Aktionen, welche aus Sensorwerten und Automaten generiert werden, das Erscheinungsbild, also der Phänotyp des Roboters. Dementsprechend ist der genotypische Suchraum die Menge aller möglichen Automaten und der phänotypische Suchraum die Menge aller Verhaltensweisen [KS08].

### 3.4 Die Simulationsumgebung

Alle in Kapitel 5 vorgestellten Simulationen wurden auf simulierten Jasmine-IIIp Robotern [KJKL08, S. 86] durchgeführt, welche, wie in Kapitel 4 erläutert, erweitert wurden. Als Simulationsprogramm wurde dabei das von Lukas König entwickelte FMG Framework genutzt. Dieses basiert, wie auch die realen Jasmine-IIIp Roboter, auf einem Evolutionären Modell, welches online und dezentral agiert und Moore Automaten nutzt, um die Aktionen der Roboter zu steuern. Online bedeutet dabei, dass die Roboter sowohl die Aufgabe haben eine gegebene Aufgabenstellung zu lösen, als auch gleichzeitig zu bewerten wie gut das aktuelle Verhalten ist. Die Evolution erfolgt also, anders als bei offline-Lernverfahren, auch „Traning Phase Evoluiton“ genannt [WGW03, S. 3], während der eigentlichen Problemlösung. Der Vorteil liegt hierbei darin, dass sich das Verhalten dynamisch an Änderungen der Problemstellung oder der Umgebung anpassen kann. Dezentrale Steuerung der Roboter bedeutet, dass sowohl die Evolution als auch die Steuerung der Roboter ausschließlich auf dem Roboter (onboard) ausgeführt werden. Somit sind lokale Sensordaten die einzige verfügbare Informationsquelle. Austausch von Informationen innerhalb der Roboter ist generell möglich, allerdings nur via Infrarot zwischen Robotern und nicht über eine zentralisierte Schnittstelle. Die Simulationen verlaufen dabei zyklisch. In einem Zyklus werden die Sensorwerte aller Roboter berechnet, sowie die atomare Aktion, welche gerade durch den aktuellen Zustand vorgegeben wird, ausgeführt. Der „move“ Befehl würde beispielsweise zu einer Vorwärtsbewegung des Roboters, ein „turn left“ Befehl analog zu einer Linksdrehung führen. Zudem wird, entsprechend der im Automaten festgelegten Bedingungen, der Folgezustand berechnet. Nach einer bestimmten Anzahl von Zyklen werden außerdem Evolutionäre Operatoren auf die Individuen der Population angewandt.

Die allgemeinen Operatoren, welche in Evolutionären Algorithmen Einsatz finden, wurden bereits in Kapitel 3.1 vorgestellt. An dieser Stelle wird auf die konkrete Umsetzung

dieser Grundprinzipien im FMG Framework eingegangen. Zuerst wird eine beliebige Anzahl Roboter in der Umgebung verteilt. Ihre Genome werden standardmäßig leer initiiert, könnten aber auch bereits Automaten enthalten, um diese weiterzuentwickeln. Alle in Kapitel 5 vorgestellten Simulationsläufe wurden mit leeren Anfangsautomaten durchgeführt. Alle 50 Zyklen<sup>2</sup> wird die Güte der Roboter evaluiert. Hierzu wird der Funktionswert der in Kapitel 4 vorgestellten Gütefunktionen berechnet und auf den aktuellen Gütefunktionswert addiert. So wird die Fitness der Individuen an die derzeitige Performanz ihres Verhaltens angepasst. Alle 300 Zyklen wird die aktuelle Güte des Roboters halbiert, um den Einfluss alter Verhaltensweisen über die Zeit immer weiter abzuschwächen. Dabei verändern sich die Verhaltensweisen durch Mutationen. Im FMG Framework sind verschiedene Mutationsoperatoren implementiert, aus denen gewählt werden kann. Die folgenden Versuche wurden alle mit derselben Mutationsart durchgeführt, welche direkt auf dem Genotyp agiert, da so in bisherigen Untersuchungen die besten Ergebnisse für komplexe Problemstellungen erzielt werden konnten. Dies ist unter anderem darauf zurückzuführen, dass die Automaten tendenziell größer werden als bei anderen Mutationsarten. Die konkrete Umsetzung ist auf Seite 88 des Artikels „Evolving Collision Avoidance on Autonomous Robots“[KS08] nachzulesen. Ein Rekombinationsoperator existiert nicht, da es in der Evolutionären Robotik, wie bereits in Kapitel 3.1.1 erwähnt, schwierig ist sinnvoll zu rekombinieren. Hierzu müsste der Genotyp in seine Chromosomen aufgeteilt werden, damit diese neu kombiniert werden können. Dies ist bei Endlichen Automaten im Allgemeinen nicht möglich. Da die Jasmine-IIIp Roboter dezentral operieren, existiert kein globaler Selektionsoperator. Stattdessen unterlaufen die Roboter alle 300 Zyklen einem sog. Reproduktionsoperator[KS08, S. 89]. Hierbei werden die  $n$  nächsten Nachbarroboter zum aktuell ausgewählten gesucht und miteinander verglichen. Der beste Roboter gibt dann sein Genom, also seinen Endlichen Automaten, sowie seinen aktuellen Gütewert an die anderen Roboter weiter. Auf diese Weise verteilen sich besser angepasste Verhaltensweisen in der Population.

---

<sup>2</sup>alle Zyklenzahlen sind als Standardwerte nach vorhergehenden Untersuchungen [KS08] zu verstehen.  
Ein Simulationslauf besteht dabei aus 300000 Zyklen





## 4 Analyse und Entwurf

### 4.1 Sensorik

Um es den Robotern zu ermöglichen Sprache zu evolvieren, muss ihnen zuerst die Fähigkeit zur Kommunikation gegeben werden. Dies soll durch akustische Signale realisiert werden. Um kommunizieren zu können, müssen zwei grundlegende Strukturen vorhanden sein. Zum einen müssen Signale gesendet werden können, zum anderen muss auch die Möglichkeit bestehen diese Signale zu empfangen und zu interpretieren. In der Natur breitet sich Schall, ausgehend vom Punkt der Entstehung, gleichmäßig in alle Richtungen aus. Der Schalldruck, also gewissermaßen die Lautstärke, folgt dabei der sogenannten „6db-Regel“ [Kar06, S. 1030]. Diese besagt, dass der Schalldruck mit Verdopplung der Distanz zwischen Emittent und Empfänger um 6db abnimmt. Die Lautstärke, im Gegensatz zum Schalldruck, ist ein Maß dafür, wie laut ein bestimmter Schall vom menschlichen Gehör empfunden wird. Dieses Empfinden hängt dabei nicht nur vom Schalldruck, sondern auch von dessen Frequenz ab [Ter98, Kap. 10.1.2]. Da in allen Experimenten der emittierte Ton die gleiche Frequenz hat und somit Lautstärke und Schalldruck proportional zueinander sind, können die beiden Begriffe in diesem Zusammenhang synonym verwendet werden. Um die Richtung einer Schallquelle zu orten, werden bei den meisten Lebewesen zwei verschiedene Informationen ausgewertet; die zeitlichen Differenzen des Eintreffens des Schalls an den Gehörgängen, sowie die unterschiedliche Lautstärke. So wird ein Geräusch, dessen Quelle rechts von einem Individuum liegt, zuerst dessen rechtes Ohr und dieses mit einer höheren Lautstärke erreichen, da der Schalldruck proportional mit der zurückgelegten Entfernung abnimmt. Schallwellen haben darüberhinaus die Eigenschaft gewisse feste Materialien wie beispielsweise Wände zu penetrieren.

In der Simulation wurden die Roboter mit einer Reihe zusätzlicher Sensoren ausgestattet, die es ihnen erlauben akustische Signale zu empfangen. Diese Sensoren sind in ihrer Form

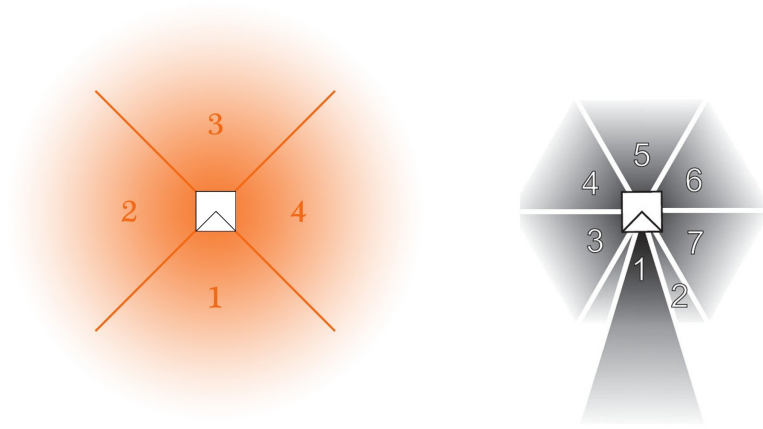


Abbildung 4.1: Beispielhafte Darstellung des Empfangsbereiches der akustischen (links) und Infrarot-Sensoren (rechts). Die Zahlen entsprechen dem Index des Sensors.

und Ausrichtung den Infrarot-Sensoren sehr ähnlich (vgl. Abbildung 4.1) und in ihrer Anzahl variabel. Damit sie immer  $360^\circ$  abdecken, wird ihre Breite automatisch auf  $\frac{2\pi}{n}$ , mit  $n$  = Anzahl der Sprach-Sensoren, gesetzt. Der Radius der Sprach-Sensoren kann vom Benutzer dabei frei gewählt werden und ist immer relativ zur Infrarot-Sensorreichweite  $\bar{s}$  anzugeben. Diese berechnet sich als Nullstelle der Funktion  $\bar{s} = v \times \log_{c_2} \frac{0.5}{c_1}$  wobei  $c_1, c_2$  an realen Jasmine-IIIp Robotern gemessene Konstanten und  $v$  die Verzerrung der aktuellen Simulation sind. Die Verzerrung dient dazu, die Roboter samt aller Sensorreichweiten innerhalb der Umgebung zu schrumpfen (bei  $v < 1$ ) oder zu strecken (bei  $v > 1$ ) und somit das Verhältnis von Roboter zu Umgebung zu verändern. Bei einer Verzerrung von  $v = 1$  gilt beispielsweise  $\bar{s} = 248$ . Die Richtungserkennung geschieht in der Simulation über den Index der Sensoren. Es empfängt also immer der Sensor, welcher dem Emittenten zugewandt ist, dessen Signal. Dabei kann durchaus jeder der Sensoren zur gleichen Zeit ein Signal empfangen.

Die Lautstärke des empfangenen Signales wird durch den Sensorwert repräsentiert. Dieser liegt im Intervall  $[0, 255]$ , welches dem Wertebereich der Infrarot-Sensoren entspricht. Ein Sensorwert von „0“ bedeutet, dass kein Signal anliegt; ein Sensorwert von „255“, dass mit maximaler Lautstärke empfangen wird, der Emittent des Schalls sich also direkt neben dem Roboter befindet.



Bisher wurde nur der Prozess des Empfangens eines akustischen Signals beschrieben, nicht aber woraus es besteht noch wie es erzeugt wird. Der Broadcast wird in der Simulation durch eine Bool'sche Variable realisiert, welche angibt ob der betreffende Roboter sendet oder nicht. Zum Starten des Broadcasts muss also diese Variable auf „wahr“ gesetzt werden. Hierzu wurde die Menge der Zustände  $Q$  ihrer Endlichen Automaten um einen weiteren Zustand „send“ erweitert. Nachdem dieser aufgerufen wird, bleibt der Broadcast für eine im Vorhinein festgelegte Anzahl an Zyklen aktiv, bevor die Bool'sche Variable wieder auf „falsch“ gesetzt wird. Der Grund für diese Realisation liegt darin, dass die Roboter zu einem bestimmten Zeitpunkt immer nur in einem Zustand sein können, sie aber trotzdem die Möglichkeit haben sollen gleichzeitig zu senden und sich zum Beispiel fortzubewegen. Dies hätte auch über zwei separate Zustände, einen zum Aktivieren und einen zum Deaktivieren des Broadcasts, gelöst werden können. Allerdings wäre so der Suchraum unnötig vergrößert worden, ohne eine wesentlich erweiterte Funktionalität zu bieten.

Der genaue Algorithmus, nach dem der Sensorwert berechnet wird, läuft wie folgt ab: Zuerst wird das Feld aller Roboter traversiert, um den nahegelegensten Roboter zu finden, der sendet und innerhalb des aktuell betrachteten Sensorbereichs liegt. Abhängig von der Anzahl der Sprachsensoren der Roboter wurden hierzu verschiedene Fälle implementiert. Falls mehr als zwei Sprachsensoren existieren, wird der Punkt-im-Polygon-Test[Bar05, S. 103] verwendet, um zu entscheiden, ob ein Roboter in dem Bereich des Sensors, dessen Sensorwert berechnet werden soll, liegt. Sind weniger als zwei Sensoren vorhanden, ist dieser Algorithmus allerdings nicht anwendbar. Zudem kann in diesem Fall der Test auf Zulässigkeit durch Methoden der analytischen Geometrie wesentlich effizienter durchgeführt werden. Wände werden bei der Suche nach zulässigen Robotern ignoriert. Der auf diese Weise gefundene Roboter darf außerdem nicht dem Roboter entsprechen, dessen Sensorwerte aktuell berechnet werden, da dies zu Folge hätte, dass sendende Roboter „taub“ wären, also keine anderen Signale empfangen könnten. Dies wäre gewissermaßen eine „Strafe“ für die sendenden Roboter, da die später in diesem Kapitel vorgestellten Aufgaben den Austausch von Signalen voraussetzen. Wenn nun sendende Roboter keine Informationen mehr empfangen können, sind sie in der Evolution benachteiligt, was potenziell dazu führen würde, dass im entwickelten Verhalten kaum Sprache enthalten wäre. Falls ein Roboter existiert, welcher die oben genannten Bedingungen erfüllt, berechnet sich der konkrete Sensorwert  $V$  gemäß der Formel  $V(x) = 255 \times \frac{r_{max}-x+|r_{max}-x|}{2 \times r_{max}}$ ,

welche äquivalent ist zu  $V(x) = 255 \times \max\{\frac{r_{max}-x}{r_{max}}, 0\}$ , wobei  $r_{max}$  der Sensorradius und  $x$  die Entfernung vom aktuellen Roboter zum Emittenten des Schallsignals ist. Die gewählte Realisation der Sensorik hat allerdings auch zur Folge, dass immer nur der nahegelegenste sendende Roboter gehört wird.

Diese Implementierung übernimmt die grundlegenden Eigenschaften von Schall in der Natur, abstrahiert diese allerdings in einigen Punkten. So wurde beispielsweise die „6db-Regel“ nicht implementiert, sondern stattdessen eine lineare Funktion gewählt. Diese beeinflusst zwar die Sensorwerte, behält jedoch die Transitivität und somit alle wichtigen Informationen. Außerdem wurde von der Implementation zusätzlicher Effekte, die bei der Ausbreitung von Schall auftreten, wie beispielsweise der Reflektion des Schalls von Wänden, dem Dopplereffekt oder Überlagerungen beziehungsweise Auslöschen zweier Schallwellen, abgesehen. Der Grund hierfür ist zum einen, dass wenn die vorgestellte Implementierung auf reale Roboter portiert würde, die emittierten Signale extrem schwach sein müssten, damit die verwendeten maximalen Reichweiten (vgl. Kapitel 5) erreicht werden. Zum anderen würden solche Effekte zu einem komplexeren Suchraum und zu einem schwerer verständlichen Verhalten führen.

Im „Futtersuche Szenario“, welches in Kapitel 4.3 vorgestellt wird, wird zudem noch ein Futtersensor benötigt. Dieser misst allgemein die Entfernung zu den, wie in Kapitel 4.4.1 vorgestellt, zufällig in der Umgebung verteilten Futterplätzen.

Ein **Futterplatz** ist dabei ein kreisförmiger Bereich innerhalb der Umgebung, welcher den Robotern gemäß der in Algorithmus 1 (Kapitel 4.3) vorgestellten Bewertungsfunktion zu einer erhöhten Güte verhilft. Das „Futter“ an diesen Futterplätzen ist dabei limitiert und nimmt immer dann, wenn mindestens ein Roboter an der entsprechenden Futterstelle ist, um eine Einheit ab. Es wurden auch Versuche mit einer zur Anzahl der Roboter proportionalen Abnahme des Futters unternommen. Allerdings führt dies zu einem in Kapitel 4.4.2 näher beschriebenen adversen Selektionsproblem der Agenten. Ist das Futter einmal aufgebraucht, wird ein neuer Futterplatz nach dem in Kapitel 4.4.1 erläuterten Verfahren platziert. Die Anzahl der Futterplätze innerhalb der Umgebung sowie deren Radius ist dabei vom Benutzer einstellbar, ihre Form ist jedoch immer kreisförmig.

Der Futtersensor (vgl. Kapitel 4.1) ist in Breite und Länge variabel, in der Standardparametrisierung jedoch schmaler und länger als beispielsweise die Sprachsensoren. Dies hat den Vorteil, dass Futterplätze auch über größere Distanzen erkannt und genauer

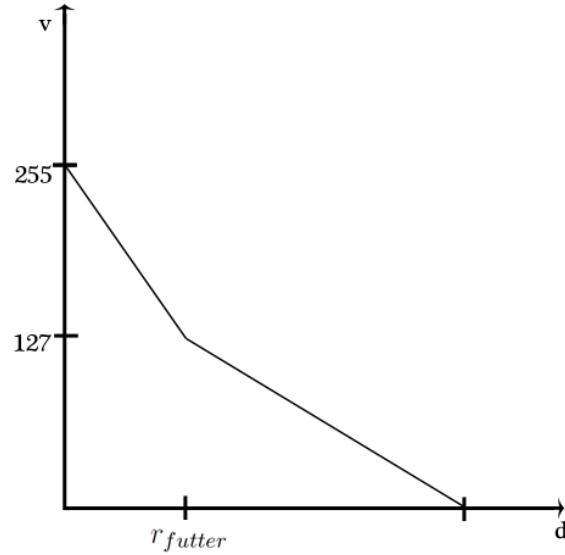


Abbildung 4.2: Sensorwert  $v$  in Abhängigkeit von der Entfernung  $d$

lokalisiert werden können. Der Sensorwertverlauf ist, anders als bei den akustischen Sensoren, nur teillinear (vgl. 4.2). Falls sich der aktuelle Roboter innerhalb der Futterquelle befindet, wird der Sensorwert  $v$  aus dem Intervall  $[128, 255]$ , abhängig von seiner Entfernung zum Mittelpunkt  $d$ , nach der Formel  $v = 255 - \frac{d \times 127}{r_{futter}}$  vergeben, wobei  $r_{futter}$  den Radius der Futterstelle bezeichnet. Der Sensorwert 128 bedeutet also, dass der Roboter genau auf dem Rand der Futterstelle steht, bei einem Sensorwert von 255 würde sich der Roboter genau in der Mitte der Futterquelle befinden. Falls der Roboter außerhalb der Futterstelle ist, nimmt der Sensor analog Werte aus dem Intervall  $[0, 127]$  an. Hierzu wird die Entfernung  $d$  zwischen dem Roboter und dem nächstgelegenen Punkt innerhalb der Futterquelle berechnet, um daraus des Sensorwert  $v = 127 - \frac{d}{r_{futter}}$  zu berechnen. Im einfachsten Fall, falls der Mittelpunkt innerhalb des Sensorbereichs liegt, beträgt die kürzeste Entfernung  $d$  zwischen dem Mittelpunkt des Roboters  $M_{rob}$  und der Futterstelle  $M_{futter}$ :  $d = ||\overrightarrow{M_{rob}M_{futter}}|| - r_{futter}$ .  $||x||$  sei dabei die Länge des Vektors  $x$ . Falls  $M_{futter}$  außerhalb des Sensorbereichs liegt, impliziert dies aber noch nicht, dass  $v = 0$  gilt, wie in 4.6 zu erkennen ist. In diesem Fall müssen die Schnittpunkte  $I_1, I_2$  der seitlichen Vektoren  $\vec{b}_{rechts}, \vec{b}_{links}$  mit dem Rand der Futterquelle berechnet werden. Die Distanz zum nächsten

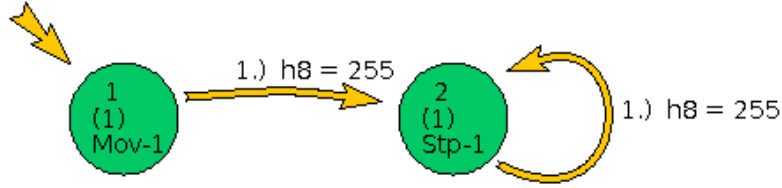


Abbildung 4.3: Beispiel für einen Automaten, welcher einen Roboter in der Umgebung umherfahren lässt bis er auf eine Futterquelle trifft und anschließend stoppt

Futterpunkt ergibt sich dann als  $d = \min\{\|\overrightarrow{M_{rob}I_1}\|, \|\overrightarrow{M_{rob}I_2}\|\}$ . Bevor diese etwas unkonventionelle Transformation der Entfernung in Sensorwerte zur Anwendung kam, existierte eine lineare Version. Diese berechnete den Abstand zum nahegelegensten Punkt, welcher zu einer Futterquelle gehörte und transformierte ihn in das Intervall  $[0, 255]$ . Ein Sensorwert von 255 bedeutete, dass die Entfernung zwischen Roboter und Futterquelle 0 betrug, der Roboter sich also auf der Futterquelle befand. Der Nachteil dieser Realisation wird anhand eines einfachen Beispiels schnell deutlich: Angenommen die Roboter sollen ein Verhalten entwickeln, bei dem sie wahrlos in der Umgebung umherfahren bis sie auf eine Futterquelle treffen und dann stehenbleiben. Ein geeigneter Automat ist in Abbildung 4.3 aufgezeichnet. ‘h8’ sei dabei der Futtersensor. Wichtig für diesen Automaten ist die Bedingung  $h8 = 255$ , welche den Roboter dazu veranlasst zu stoppen, sobald er sich auf einer Futterquelle befindet. Da alle Automaten durch Evolution, also mehr oder weniger durch Zufall entstehen, ist eine solche Parametrisierung unwahrscheinlich. Sobald der Schwellenwert ungleich 255 ist, wird der Roboter nicht mehr anhalten. Im teillinearen Fall steht für die Parametrisierung ein wesentlich breiteres Intervall von  $[128, 255]$  zur Verfügung. Zwar würde ein Roboter, welcher die Bedingungen  $h8 \geq 200$  beinhaltet, nicht direkt am Rand der Futterquelle anhalten, jedoch spätestens auf dem halben Weg vom Rand der Futterquelle zum Mittelpunkt stoppen. Auf dieser Basis kann sich das Verhalten besser weiterentwickeln als im linearen Fall. Zwar könnte der Sensorwert durchaus auf andere Weise und trotzdem auf Basis einer Lineartransformation berechnet werden, indem nicht der Abstand zum nahegelegensten Futterpunkt, sondern zum Futtermittelpunkt berechnet würde. Allerdings ist in diesem Szenario der Sensorwert, ab dem sich der Roboter

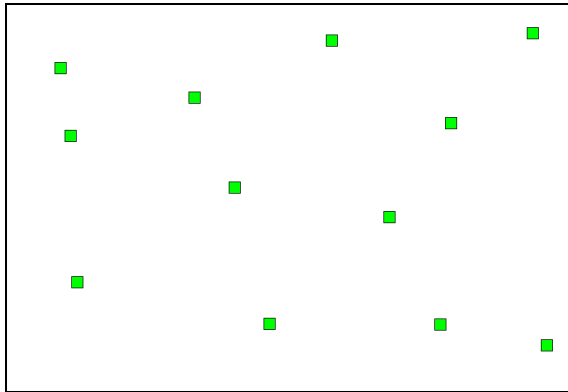


Abbildung 4.4: „Soziale Umgebung“ mit „Dummy-Robotern“

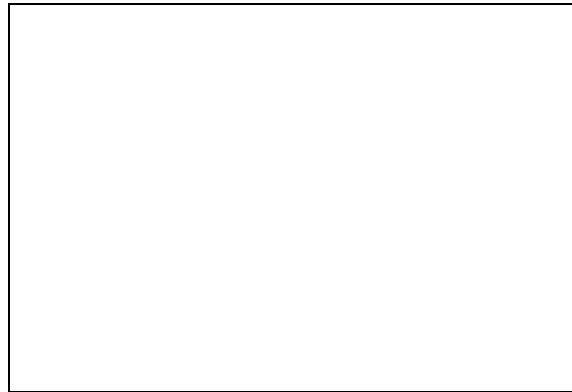


Abbildung 4.5: Leere Umgebung.

innerhalb der Futterquelle befindet, abhängig von der Breite der Futterquelle. Da hierunter aber die Portierbarkeit der Lösung in andere Szenarien leidet, wurde für alle weiteren Versuche die teillineare Sensorberechnung benutzt.

## 4.2 „Soziales Szenario“

Die im vorhergehenden Kapitel 4.1 beschriebenen Grundlagen zur Entwicklung von Sprache werden zuerst in dem sogenannten „Sozialen Szenario“ getestet. Dieses ist an das aus der Tierwelt bekannte Herdenverhalten angelehnt. So werden die Roboter belohnt, wenn sie in einer Gruppe in der Umgebung umherfahren. In einer Gruppe bedeutet dabei, dass mindestens ein Roboter in der Nähe eines anderen sein muss, damit die beiden als Gruppe gelten. Eine Belohnung proportional zur Größe der Gruppe wurde nicht implementiert, da dies dazu führen würde, dass der Bewegungsbonus in Relation viel zu klein ausfallen würde. Der Bewegungsbonus wird benutzt, um das lokale Optimum zu umgehen, bei dem sich die Individuen einmal einander annähern und sich danach nicht wieder bewegen. Gleichzeitig werden die Roboter bestraft, wenn sie sich in der Nähe mindestens einer Wand befinden. Der Radius um den Roboter, in dem er für andere Roboter belohnt, respektive für Wände bestraft wird, ist gleich groß und vom Benutzer frei wählbar. Generell ist es aber sinnvoll, diesen kleiner als den Sende- und Infrarot-Sensorradius zu wählen, da die Roboter sonst für etwas bestraft oder belohnt werden, was sie nicht wahrnehmen können.

In einer einfach rechteckigen Umgebung würde ebenfalls ein Verhalten, in welchem die Roboter nur mittig im Kreis fahren, zu der gewünschten Verhaltensweise führen. Deshalb wurde eine neue Umgebung entwickelt, in welcher weitere Wände wahllos verteilt sind, die in Form und Größe Robotern ähneln.

Der Sinn dieses Szenarios ist, dass die Roboter mit ihren Infrarot-Sensoren nur die Entfernung zu einem Objekt, nicht aber dessen Beschaffenheit erfassen können. Somit sind im ersten Schritt Roboter nicht von Wänden zu unterscheiden. Für diese Differenzierung wird es also notwendig zu kommunizieren und Sprache zu entwickeln.

### **4.3 „Futtersuche Szenario“**

Das zweite Szenario in dem untersucht wurde, ob Roboter Sprache entwickeln, war das sogenannte „Futtersuche Szenario“, welches den Schwerpunkt dieser Arbeit darstellt. Aufgabe der Roboter ist es hierbei, sich auf einer Futterstelle aufzuhalten. Für das Auffinden der Futterquelle dient der in Kapitel 4.1 eingeführte Futtersensor. Ähnlich wie bei Ameisen ist es auch für die Roboter vorteilhaft, wenn sich mehrere Individuen an der Futterquelle befinden. Bei Ameisen dient dies der Verteidigung der Futterstelle gegenüber rivalisierenden Ameisenkolonien oder Fressfeinden sowie des schnelleren Abtransportes der Nahrung zum Bau. Dieser Grundgedanke wurde auch auf die Roboter angewandt, indem sie für dieses Verhalten belohnt werden. Eine Erweiterung hinsichtlich des Abtransports des Futters ist denkbar, allerdings für einen ersten Versuch zu komplex.

Anders als im „Sozialen Szenario“ ist die Aufgabe der Futtersuche auch ohne Sprache lösbar. Kann der Futtersensor die gesamte Umgebung überblicken, besteht keine Notwendigkeit Sprache zu entwickeln, da jedes Individuum eine Futterquelle finden kann. Sobald der Futtersensor jedoch eine geringe Reichweite hat, sollte ein Individuum, welches eine Futterquelle gefunden hat, dem Rest der Population diese Information mitteilen, um die volle Anzahl an Boni (vgl. Tabelle 5.1) zu erhalten. Die Evolution von Sprache im „Futtersuche Szenario“ ist aber nicht nur aufgrund der anderen Lösbarkeit der Aufgabe wesentlich komplexer. Während im „Sozialen Szenario“ ein dauerhaftes Senden das Lösen der Aufgabe ermöglicht, ist in diesem Szenario ein ständiges Broadcasten nicht sinnvoll. Vielmehr darf ein Individuum nur dann senden, wenn es sich auf einer Futterquelle be-

findet. Parallel muss sich jedoch ebenfalls das Verhalten entwickeln auf einen sendenden Roboter zuzufahren.

Die Bewertungsfunktion des „Futtersuche Szenarios“ besteht aus zwei weiteren Komponenten, mit denen die zu lösende Aufgabe weitergehend verändert oder die Evolution von Sprache erleichtert werden kann. Zum einen besteht die Möglichkeit Roboter zu belohnen, falls sie sich in Bewegung befinden. Hiermit könnte eine anfängliche Startschwierigkeit überwunden sowie das optimale Verhalten komplexer gestaltet werden, da die Individuen dann auf der Futterstelle fahren müssten, anstatt nur auf ihr zu stehen. Zum anderen können Roboter, welche sich nicht auf einer Futterquelle befinden, bestraft werden. Dies ist zwar auf den ersten Blick der gleiche Effekt wie sie für die Futtersuche zu belohnen, allerdings bewirkt die Bestrafung eine schnellere Abnahme der Güte. Ohne sie würde die Güte der Roboter nur langsam evaporieren, da sie nur relativ selten halbiert wird (vgl. Kapitel 5). Der aus diesen Einzelteilen resultierende Algorithmus zur Güteberechnung eines Individuums ist in Algorithmus 1 beschrieben.

### Algorithmus 1

(\* Berechnet die Güte eines Individuums \*)

**Input:** \* Roboter  $r$ , RoboterArray  $andereRoboter$ , Futterstellen  $fs$ , Futterbonus  $B_f$ , AndereRoboterBonus  $B_r$ , BewegungsBonus  $B_b$ , FutterStrafe  $S_f$

**Output:** Güte  $g$

1.  $g = 0$
2. **if**  $Position(r)$  ist in  $fs$
3.     **then** erhöhe  $g$  um  $B_f$
4.      $fs_i \leftarrow$  Futterstelle an der sich  $r$  befindet
5.     **for** each  $Roboter \in andereRoboter$  \*\*
6.         **if**  $Position(Roboter)$  ist in  $fs_i$
7.             **then** erhöhe  $g$  um  $B_f$
8.     **else** erniedrige  $g$  um  $S_f$
9. **if**  $Aktion(r)$  ist „MOV“
10.     **then** erhöhe  $g$  um  $B_b$
11. **return**  $g$

\* Teile der Bewertungsfunktion können deaktiviert werden, indem die entsprechenden Parameter auf den Wert 0 gesetzt werden.

\*\* Diese Realisation sorgt dafür, dass die Güte des Roboters proportional zur Anzahl der Roboter an derselben Futterstelle ist. Falls dies nicht erwünscht ist, können Zeilen 5. und 6. durch

$$if \exists Roboter \in andereRoboter | Position(Roboter) ist in fs_i$$

ersetzt werden.

## 4.4 Sonstige Implementierungen

In den folgenden Unterkapiteln wird nun die konkrete Realisation einiger spezieller Operatoren näher beschrieben. Diese sind zwar für die Durchführung der Versuche unerlässlich, nicht aber für das Verständnis der Ergebnisse.



### 4.4.1 Platzieren der Futterplätze

Wie bereits in Kapitel 4.1 erwähnt, müssen im „Futtersuche Szenario“ Futterplätze in der Umgebung verteilt werden. Dies geschieht zwar zufällig, allerdings wird dabei versucht, die Futterstelle nicht direkt unter einen Roboter zu platzieren. Dies hätte zur Folge, dass der betreffende Roboter auch ohne jede Form von Verhalten eine erhöhte Güte haben würde, was speziell zu Beginn der Simulation den Entwicklungsprozess nachteilig beeinflussen kann. Hierzu wurde ein eigener Algorithmus entwickelt, der im Folgenden vorgestellt und in Kapitel 5.4 evaluiert wird.

#### Algorithmus 2

(\* Platziert einen Futterplatz auf einer freien Stelle innerhalb der Umgebung \*)

**Input:** FutterRadius  $r$ , RoboterArray  $andereRoboter$ , Schwellenwert  $s$ , MittelpunkteDerFutterstellen  $M_{futter}$

1. Liste  $l \leftarrow$  alle Elemente  $\in andereRoboter \cup M_{futter}$
2. Punkt  $P \leftarrow$  zufälliges Element  $\in l$
3. Liste  $vList \leftarrow$  normierte Vektoren zwischen  $P$  und jedem Element  $\in l \setminus P$  welche nicht länger als  $s$  entfernt sind
4. Ordne Vektoren nach Richtung im Uhrzeigersinn, beginnend bei  $(-1, 0)$
5. Liste  $w \leftarrow$  Winkel zwischen allen Vektoren  $\in v$
6. **for**  $max\{w\}$  downto  $min\{w\}$   
 (\* Testen, ob Futterstelle zwischen  $w_i$  und  $w_j$  passt \*)
7.  $w_i, w_j$  seien die Vektoren, welche aktuellen Winkel  $w$  bilden
8. Vektor  $v \leftarrow \frac{w_i + w_j}{2}$
9. Punkt  $I \leftarrow$  Schnittpunkt der Gerade „ $P + \eta v$ “ mit einer der Spielfeldgrenzen
10. **if**  $\|\vec{IP}\| \geq s$
11.     **then** Setze  $I$  so, dass  $\|\vec{IP}\| = s$  und  $\exists \eta | I = P + \eta v$  gilt
12. Bedingung  $b_1 \leftarrow \|\vec{IP}\| \geq 2 \times r$
13. Bedingung  $b_2 \leftarrow \max\{\frac{2r}{\tan \frac{\varphi}{2}}, 2r\} := minLaenge < \|\vec{IP}\| + r$
14. **if**  $b_1 \wedge b_2 == true$
15.     **then**
16.         erzeuge neue Futterstelle mit zufälligem Mittelpunkt  $M = P + \lambda v$   
        wobei  $\lambda \in \mathbb{R}$  sodass gilt  $\|M\| \in [minLaenge, \|\vec{IP}\| - r]$
17.     **return**
18. **end**

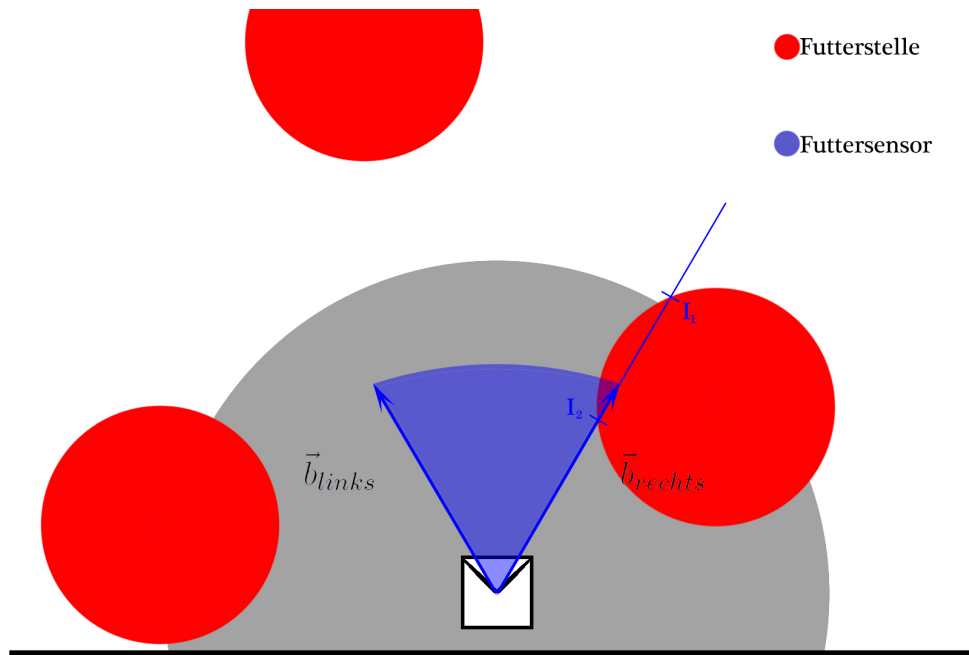


Abbildung 4.6: Beispielhafte Visualisierung der Problematik des Futterplatzierens

Der Algorithmus wählt in Schritt 2 einen Punkt  $P$  der Menge der Futterstellen und Roboter. Anschließend werden Vektoren zu allen Futterstellen und Robotern in einer begrenzten Umgebung um den ausgewählten Punkt aufgespannt. Diese werden anschließend im Uhrzeigersinn geordnet, um anschließend die Winkel zwischen ihnen zu bestimmen. Nun wird immer für zwei Vektoren, welche den aktuell betrachteten Winkel bilden, darauf getestet, ob eine Futterstelle zwischen ihnen gesetzt werden kann. Dabei wird die Liste der Winkel nach abnehmender Größe durchgearbeitet, da die Wahrscheinlichkeit die Futterquelle setzen zu können mit der Winkelgröße abnimmt. Die Kriterien nach denen entschieden wird, ob eine Futterstelle platziert werden kann, sind dabei zum einen, dass innerhalb des Winkels die Entfernung zur nächsten Wand größer als der Futterdurchmesser ist ( $b_1$ ). Zum anderen muss der Abstand zwischen den beiden umschließenden Vektoren, abhängig von dem Abstand zum Punkt  $P$ , breiter als der Durchmesser der Futterquelle sein ( $b_2$ ). Hier wird deutlich, warum bei der Betrachtung in Schritt 3 nur innerhalb der Umgebung von  $s$  um den ausgewählten Punkt gesucht wird. Würde die gesamte Umgebung betrachtet, so würde der Algorithmus kaum Plätze für die Futterquellen finden, da auch Hindernisse betrachtet werden, die aufgrund ihrer Entfernung irrelevant sind. Sind beide Bedingungen

erfüllt, wird die Futterstelle platziert. Dies bedeutet, dass der neu ermittelte Mittelpunkt der Futterquelle in die Liste aller Futterquellen eingetragen wird.

Als Folge der zweiten Bedingung platziert der Algorithmus nur Futterstellen, bei denen sichergestellt ist, dass diese überschneidungsfrei gesetzt werden können. Dabei wird in Kauf genommen, dass eventuell keine Futterquelle platziert wird, obwohl dies theoretisch möglich wäre. Dies ist darin begründet, dass bei der Prüfung auf ausreichende Breite immer der Sicherheitsabstand von einem Futterradius um die Vektoren, welche den betrachteten Winkel einschließen, eingehalten wird, auch wenn an dieser Stelle keine Futterquelle existiert. Aus diesem Grund oder weil um den anfänglich gewählten Punkt kein Platz ist, wird der Vorgang solange wiederholt, bis eine Futterquelle platziert werden konnte. Da es aber auch durchaus passieren kann, dass in der Umgebung kein Platz für eine zusätzliche Futterstelle ist, wird spätestens nach 300 Versuchen abgebrochen und eine Futterquelle rein zufällig gesetzt. Dabei werden auch Überschneidungen mit Robotern oder anderen Futterquellen in Kauf genommen, damit zumindest die Anzahl an Futterquellen während der kompletten Simulation gleich bleibt.

#### **4.4.2 Abnahme des Futters**

Die Futterquellen stellen in der Simulation nur eine begrenzte Menge Futter zur Verfügung. Dieses nimmt jeden Zyklus um eine bestimmte Anzahl Einheiten ab. Wie bereits eingangs erwähnt, wurde zunächst mit einer Verringerung der Futtermenge, welche proportional zur Anzahl der auf der jeweiligen Futterquelle befindlichen Roboter ist, experimentiert. Dies ist zwar näher an die Realität angelehnt, allerdings existiert auch ein Nebeneffekt, welcher die Evolution potenziell negativ beeinflusst. Die Roboter sollen, so die Idee des „Futtersuche Szenarios“, der restlichen Population den Fundort einer Futterstelle durch akustische Signale mitteilen. Der Anreiz hierfür liegt in der Gütefunktion, welche Robotern eine höhere Güte zuweist, wenn sie sich zusammen mit anderen auf einer Futterstelle befinden. Würde nun das Futter proportional mit der Anzahl der auf einer Futterstelle befindlichen Roboter abnehmen, würden sich die Roboter, welche andere Individuen über die Position der Futterquelle informieren, selbst bestrafen, da sie dann nur eine stark verkürzte Zeit selbst von dem Futter profitieren würden. Um diesen Effekt durch die Gütefunktion zu kompensieren, müsste der Bonus für andere Roboter an der Futterquelle  $B_r$  doppelt so groß sein wie die Summe aus  $B_f$  und  $S_f$ , da die Roboter nicht

nur für die fehlende Zeit, sondern auch für den Malus, welchen sie durch den Wegfall der Futterstelle erhalten, entschädigt werden. Dies gilt allerdings nur, falls sich nur ein weiterer Roboter an der Futterquelle befindet. Für  $n$  Roboter müsste also  $B_r \geq n \times (B_f + S_f)$  gelten, damit die Roboter keinen Nachteil durch die Kommunikation haben<sup>1</sup>. Ein solch altruistisches Verhalten<sup>2</sup> ist selbst in der Natur selten zu beobachten, meist nur in Gemeinschaften mit engen verwandschaftlichen Beziehungen [Len03, S. 113]. Um die Evolution von Sprache nicht weiter zu erschweren, wird das Futter der Futterstelle, falls diese in Benutzung ist, unabhängig von der Anzahl der auf ihr befindlichen Individuen konstant um eine Einheit reduziert.

### 4.4.3 Visualisierung

Um die durchgeführten Versuche nachvollziehen zu können, musste die Möglichkeit geschaffen werden, die Futterstellen im FMG Framework anzeigen zu lassen. Hierzu musste sowohl die graphische Oberfläche, als auch die Aufnahmen des „Tracebetrachters“ angepasst werden. In einem ersten Schritt wurde die existierende GUI dahingehend angepasst, dass während der Simulation Futterplätze angezeigt werden. Die existierenden Methoden wurden so erweitert, dass die Pixel innerhalb des Futterradius um die Mittelpunkte der Futterstellen eingefärbt werden. Als Farbe der Futterplätze wurde rot gewählt. Anfänglich wurde die Speicherung der Futterquellen über ein Feld gelöst, welches genau so viele Elemente besaß wie die Umgebung Pixel und die Einträge die Anzahl Futter pro Pixel angaben. Diese Realisation war jedoch sehr langsam, weshalb diese Lösung durch die Speicherung der Mittelpunkte ersetzt wurde.

Darüber hinaus müssen Futterstellen auch dann sichtbar sein, wenn die Aufnahmen eines Versuches betrachtet werden. Hierzu muss während der Simulation zu jedem Snapshot auch die Position der Futterstellen, also deren Mittelpunkte gespeichert werden. Dies geschieht über eine einfache Textdatei, welche in jeder Zeile jeweils die Futterquellen eines Snapshots durch Semikola getrennt, enthält. Die x- und y-Koordinaten der Mittelpunkte werden wiederum durch Kommata voneinander getrennt. Wird die Aufnahme geladen, so

---

<sup>1</sup>Dies gilt nur, falls die Höhe der Belohnung unabhängig von der genauen Anzahl der zusätzlichen Roboter an der entsprechenden Futterstelle ist. Trifft dies nicht zu, muss für den Bonus  $B_r$  gelten  $B_r \geq (B_f + S_f)$ .

<sup>2</sup>Altruistisches Verhalten: Eine „selbstlose“ Verhaltensweise, welche ein Individuum mehr kostet als es ihm einbringt. Dies geschieht zum Nutzen anderer Individuen.

wird auch die entsprechende Textdatei, falls vorhanden, eingelesen und die Darstellung verläuft analog zu der der graphischen Oberfläche.



# 5 Evaluation

## 5.1 Methodik

Die in dieser Arbeit durchgeführten Versuche lassen sich thematisch in zwei Bereiche gliedern. Der kleinere Teil beschäftigt sich mit der weiteren Untersuchung des „Sozialen Szenarios“ während sich der größere Teil mit dem „Futtersuche Szenario“ befasst.

In beiden Fällen werden Simulationen mit Hilfe des FMG Frameworks auf Poolrechnern des Institutes für Angewandte Informatik und Formale Beschreibungsverfahren (AIFB) durchgeführt. Während der Simulationen wird alle 500 Zyklen eine Momentaufnahme gespeichert. Diese enthält alle relevanten Informationen der Simulation zum gegebenen Zeitpunkt, beispielsweise die Position der Roboter, ihre Automaten oder die Positionen der einzelnen Futterquellen. Aus diesen Daten kann der Verlauf der Evolution rekonstruiert und das Verhalten der Roboter analysiert werden. Der Standardparametersatz, welcher für die Simulationen verwendet wurde, ist in Kapitel 7.2 dargestellt.

### 5.1.1 „Soziales Szenario“

In vorherigen Untersuchungen [WK10] wurde bereits gezeigt, dass Roboter in der Lage sind im „Sozialen Szenario“ Sprache zu entwickeln. In seltenen Fällen zeigen sie ein Verhalten, welches sich nicht ohne den Einsatz von Sprache erklären lässt. Dabei vermeiden die Roboter sowohl den Bereich am Rand der Umgebung als auch den Bereich um die in der Umgebung verteilten „Dummy-Roboter“, in welchen sie bestraft werden. Andererseits suchen sie die Nähe anderer Individuen und fahren mit diesen in der Umgebung umher, wie die Trajektorien in Abbildung 5.1 zeigen. Das Verhalten enthält somit alle Aspekte der Gütefunktion. Die Analyse der Automaten ergab, dass das zugrunde liegende Verhalten auf regelmäßigen kreisenden Bewegungen basiert, bei denen ein Roboter nur dann auf ein

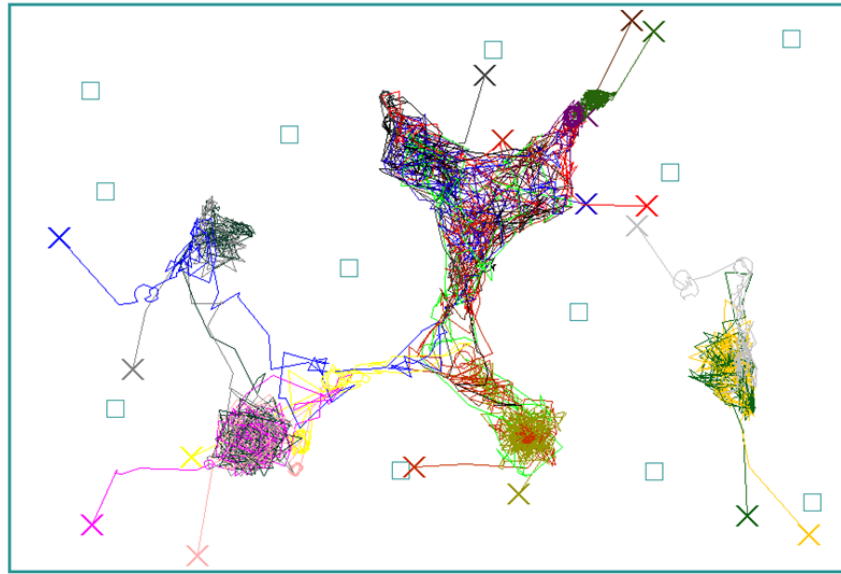


Abbildung 5.1: Trajektorien eines Roboterschwarms, welcher sowohl Wände als auch „Dummy-Roboter“ vermeidet.

Hindernis, welches er mit Hilfe der Infrarot-Sensoren erfasst, zuführt, wenn er aus dieser Richtung auch ein Broadcast Signal empfängt.

Um sicherzustellen, dass dieses Verhalten nicht auf einem anderen Effekt beruht, werden in dieser Arbeit weitere Tests durchgeführt. Theoretisch ist es möglich, dass ein reines „Collision-Avoidance“ Verhalten ähnliche Resultate zeigt, da Roboter ein Infrarotsignal wesentlich schlechter reflektieren als Wände. Auf diese Weise erhält ein Roboter, welcher vor einer Wand steht, einen deutlich höheren Sensorwert, als wenn er in gleicher Entfernung vor einem anderen Roboter steht. Auf diese Weise könnte ein „Collision-Avoidance“ Verhalten, bei dem sich Roboter verschiedenen Objekten bis zu einem gewissen Sensorwert annähern, zu ähnlichen Resultaten führen. Um diese Theorie zu validieren oder zu widerlegen, werden in dieser Arbeit zusätzliche „Taub“ Tests durchgeführt, bei denen die Roboter der Population wieder in der „Sozialen Umgebung“ (vgl. Abb. 4.4) verteilt und nach der „Sozialen Gütefunktion“ bewertet werden. Allerdings wird ihnen dabei nicht die Möglichkeit gegeben zu kommunizieren.

Weiterhin wurde in den bisherigen Untersuchungen der Einfluss verschiedener Parameter noch nicht getestet. Im Weiteren soll speziell der Einfluss des Broadcast-Radius weiter untersucht werden.



### 5.1.2 „Futtersuche Szenario“

Anders als im „Sozialen Szenario“ wurden im „Futtersuche Szenario“ bisher noch keine Untersuchungen durchgeführt. Somit werden die ersten Tests mit dem Standardparametersatz durchgeführt, um die Parameter aus den so gewonnenen Erkenntnissen inkrementell zu verändern, mit dem Ziel die Evolution des gewünschte Verhaltens zu ermöglichen. Für die nicht im Standardparametersatz enthaltenen Parameter wurden folgende Parameter verwandt:

Tabelle 5.1: Erweiterter Parametersatz\*

Parameter	Wert	Bedeutung
anzahlOhren	4	Anzahl der Sprachsensoren**
broadcastRadius	6	Radius über den hinweg Broadcast empfangen werden kann
broadcastDauer	10	Dauer eines Broadcasts**
Futterradius	150	Radius der Futterquelle
Futtersensorradius	400	Radius des Futersensors
Futtersensorbreite	0.5	Breite des Futersensors in Bruchteilen von $\pi$
Futtermenge	250	Menge an verfügbarem Futter pro Futterquelle
Futterquellenanzahl	1	Anzahl der Futterquellen
andereRoboterBonus	0	Bonus, der für die Existenz anderer Roboter an der gleichen Futterstelle gegeben wird
Futterbonus	2	Bonus, der gegeben wird, falls sich der Roboter an einer Futterstelle befindet
Bewegungsbonus	0	Bonus, falls sich der Roboter bewegt

\* Alle nicht aufgeführten, aber trotzdem relevanten Parameter sind im Standardparametersatz in Kapitel 7.2 zu finden.

\*\* Parameterwerte haben sich in vorherigen Untersuchungen als vorteilhaft erwiesen

### 5.1.3 Auswertung & Gütemaß

Bevor im weiteren Verlauf dieses Kapitels auf die genauen Ergebnisse der Simulationen eingegangen wird, wird an dieser Stelle zuerst der Prozess der Analyse der Simulationen näher beschrieben. Wie bereits eingangs erwähnt, sind die während der Simulation aufgenommenen Momentaufnahmen Grundlage der Analyse. Die genaue Rekonstruktion eines Laufes ist sehr zeitintensiv. Um nicht unnötig viel Zeit für die Analyse von Läufen

aufzuwenden, in welchen ein schlecht an die Umgebung angepasstes Verhalten evolviert wurde, werden zu Beginn des Analyseprozesses möglichst viele solcher Simulationen ausgewählt.

Das erste Hilfsmittel, mit dem sich entscheiden lässt, wie gut ein evolviertes Verhalten angepasst ist, ist die Gütefunktion der Population. Zu jedem Zeitpunkt der Simulation an dem die Güte der Roboter berechnet wird, wird diese gespeichert und am Ende der Simulation in einem Schaubild dargestellt. Dieses beinhaltet nicht nur die einzelnen Gütwerte, sondern ebenso die durchschnittliche Güte der gesamten Population zu jedem Zeitpunkt. Diese ist jedoch starken Fluktuationen unterworfen. Zur besseren Analysierbarkeit ist deshalb ebenfalls der gleitende Durchschnitt der durchschnittlichen Güte eingezeichnet.

Anhand des gleitenden Durchschnitts lassen sich nun verschiedene Informationen ablesen. Zum einen ist der Absolutwert der Güte bereits ein erster Indikator, wie gut das Verhalten der Population ist und kann, wie später in Kapitel 5.3 gezeigt, auch verschiedene Verhaltensweisen charakterisieren. Allerdings ist der Wert stark von der Umgebung abhängig, also beispielsweise bei gleichem Verhalten in einer Umgebung mit mehr Futterquellen deutlich höher als in einer Umgebung, in der sich weniger Futterstellen befinden.

Ein deutlich zuverlässiger Indikator für den Verlauf der Evolution ist die Entwicklung des gleitenden Durchschnitts. Da die Automaten der Individuen zu Anfang jeder Simulation leer initiiert werden, gibt der anfängliche Gütwert gewissermaßen die Nulllinie an. Entwickelt sich ein förderliches Verhalten innerhalb der Population, steigt die Güte der Individuen. Da Individuen mit höherer Güte ihre Automaten relativ schnell in der Population weitergeben, setzt sich der Anstieg der Güte relativ schnell im gleitenden Durchschnitt durch. Ist ein solcher Anstieg zu beobachten, sollte der entsprechende Simulationslauf genauer analysiert werden. Entwickelt sich der gleitende Durchschnitt allerdings dauerhaft negativ, ist dies ein guter Indikator für ein Verhalten, welches nicht gut an die Umgebung angepasst ist und deshalb keine weiteren Betrachtungen erfordert.

Diese Klassifizierungen sind jedoch keineswegs statisch zu betrachten. So kann es speziell gegen Ende des Güte-Graphen dazu kommen, dass Verhalten entwickelt wurde, dies aber nicht mehr im Graphen angezeigt wird, da dieser 50000 Zyklen vor Schluss der Simulation endet, weil anschließend keine Datenpunkte mehr zur Berechnung zur Verfügung stehen. Sollte sich also gegen Ende des Graphen ein Aufwärtstrend des gleitenden Durchschnitts abzeichnen, kann es durchaus lohnenswert sein, die entsprechende Simulation genauer zu

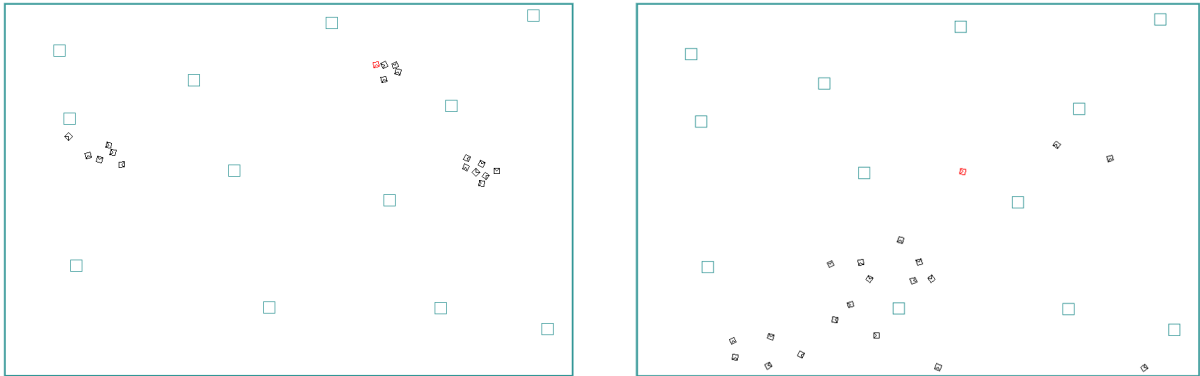


Abbildung 5.2: Vergleich zweier Momentaufnahmen von Populationen, welche Sprache nutzen (links) oder ein Verhalten ohne Sprache entwickelt haben (rechts)

betrachten. Außerdem kann es speziell im „Sozialen Szenario“ zu sehr langsamen Gütewertverbesserungen innerhalb der Population kommen, sodass kein Sprung des gleitenden Durchschnittes zu beobachten ist. Außerdem sollten exemplarisch auch schlechtere Verhaltensweisen analysiert werden, um dadurch Rückschlüsse auf mögliche Schwierigkeiten oder unvorteilhafte Parametrisierungen schließen zu können.

Um mögliche Fehlklassifikationen so gering wie möglich zu halten, werden weitere Analysemechanismen angewandt. Hierzu können beispielsweise die Trajektorien der Aufzeichnungen auf systematisches Verhalten untersucht werden, im „Sozialen Szenario“ beispielsweise auf Vermeiden von Hindernissen. In diesem Szenario ist darüber hinaus auch der Abstand der einzelnen Individuen untereinander ein guter Indikator für die Qualität eines Verhaltens. Wie Abbildung 5.2 zeigt, ist der Abstand der Individuen zueinander, welche Kommunikation nutzen, deutlich geringer als bei jenen, die ein Verhalten ohne Kommunikation entwickelt haben. Im „Futtersuche Szenario“ können zudem aus der Anzahl der geleerten Futterplätze weitere Informationen gezogen werden. Hierzu werden die bei jeder Momentaufnahme gespeicherten Positionen der Futterquellen verglichen. Die so ermittelten Abweichungen werden in einem Graph aufgetragen, wobei die Winkelhalbierende die maximal mögliche Anzahl an Futterquellenänderungen angibt, um das Schaubild zu normieren. Ein solcher Graph ist in Abbildung 5.3 dargestellt. Auch hierbei ist zu beachten, dass beispielsweise einfaches Umherfahren in einer Umgebung mit fünf Futterquellen mehr Futterquellen zufällig geleert werden, als in einer Umgebung, welche zum Beispiel nur eine Futterquellen enthält.

Mit Hilfe dieser zusätzlichen Methoden kann die Menge der zu betrachtenden Simulatio-

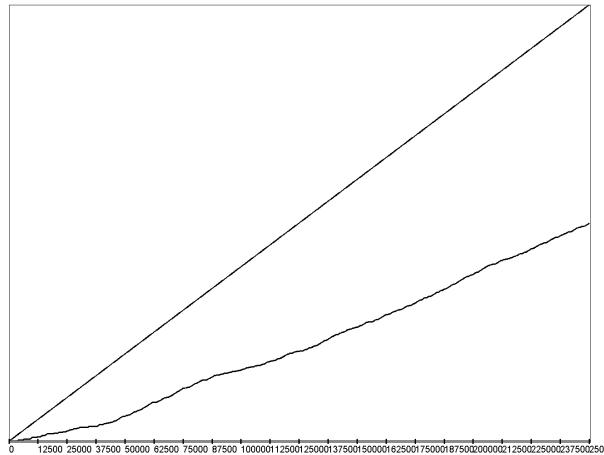


Abbildung 5.3: Beispiel eines Schaubildes, welches die Anzahl der geleerten Futterplätze (unterer Graph) der maximal möglichen Anzahl der zu leerenden Futterplätze (Winkelhalbierende) gegenüberstellt.

nen reduziert werden, ohne jedoch ein zu hohes Risiko einzugehen gute Verhaltensweisen zu übersehen. So dienen die bisher vorgestellten Mechanismen lediglich dazu, eine Vorauswahl zu treffen und potenziell wenig angepasste Verhaltensweisen auszusortieren. Der endgültige Schritt, welcher zur Klassifizierung eines Verhaltens führt, ist der „Live“ Test. Erst wenn sich hier zeigt, dass das beobachtete Verhalten nicht ohne Sprache zu erklären ist, wird der Simulationslauf als erfolgreich gewertet.

## 5.2 Ergebnisse „Soziales Szenario“

Im folgenden Kapitel werden die Ergebnisse der ergänzenden Tests, welche im „Sozialen Szenario“ durchgeführt wurden, zusammengefasst.

### 5.2.1 „Taub“ Test

Wie bereits eingangs erwähnt, werden bei den sogenannten „Taub“ Tests Roboter in der „Sozialen Umgebung“ verteilt und ihnen die Aufgabe gestellt Wände zu vermeiden und die Nähe anderer Roboter zu suchen (vgl. Kapitel 4.2). Allerdings wird ihnen die Möglichkeit genommen zu kommunizieren, indem die Anzahl ihrer Sprachsensoren auf 0 gesetzt

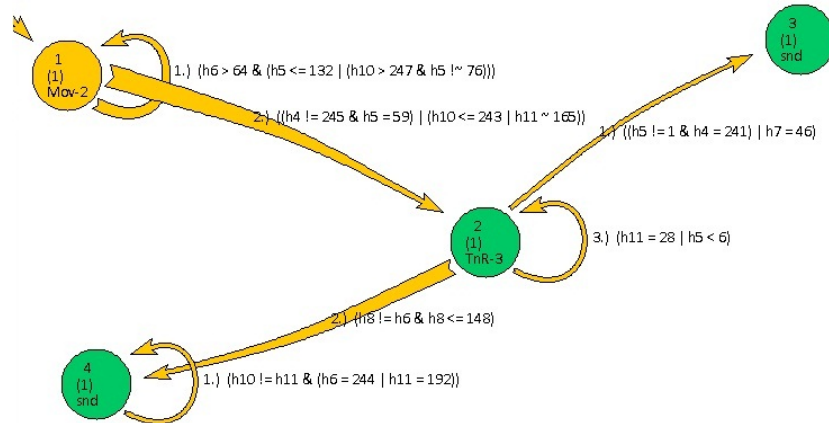


Abbildung 5.4: Automat, welcher das in Kapitel 5.2.2 beschriebene Verhalten erzeugt.

wird. So soll überprüft werden, ob die in vorherigen Untersuchungen festgestellten „Wall-Avoid“ Verhaltensweisen [WK10] eventuell nur ein Spezialfall des „Collision Avoidance“ Verhaltens sind.

Bei den durchgeführten Simulationen zeigen die Roboter keine Anzeichen für ein ähnliches Verhalten, sondern fahren entweder zufällig in der Umgebung umher oder drehen sich in regelmäßigen Kreisen auf der Stelle. Diese Ergebnisse unterstützen die These, dass im „Wall-Avoid“ Verhalten Sprache evolviert wurde.

## 5.2.2 Broadcast Dauer

In den bisherigen Untersuchungen des „Sozialen Szenarios“ wurde nur mit einer Broadcast-Dauer von zehn Zyklen getestet. Als Teil dieser Arbeit wurden die Tests auch auf kürzere Zeiträume von einem, zwei und fünf Zyklen ausgeweitet. Ähnlich wie bei den „Taub“ Tests entwickeln sich Verhaltensweisen, bei denen die Individuen sich entweder in regelmäßigen Kreisen drehen oder zufällig umherfahren. Zusätzlich tritt während einer Simulation mit der Broadcast-Dauer von einem Zyklus auch eine Verhaltensweise auf, die dem „Wall Avoid“ Verhalten ähnelt, welches schon aus bisherigen Untersuchungen [WK10] bekannt ist. Der zugehörige Automat und die von ihm hervorgerufenen Trajektorien sind in den Abbildungen 5.4 sowie 5.5 dargestellt. Auch dieses Verhalten beruht darauf, dass sich die Individuen um die eigene Achse drehen und nur dann auf ein Objekt zufahren, wenn dieses

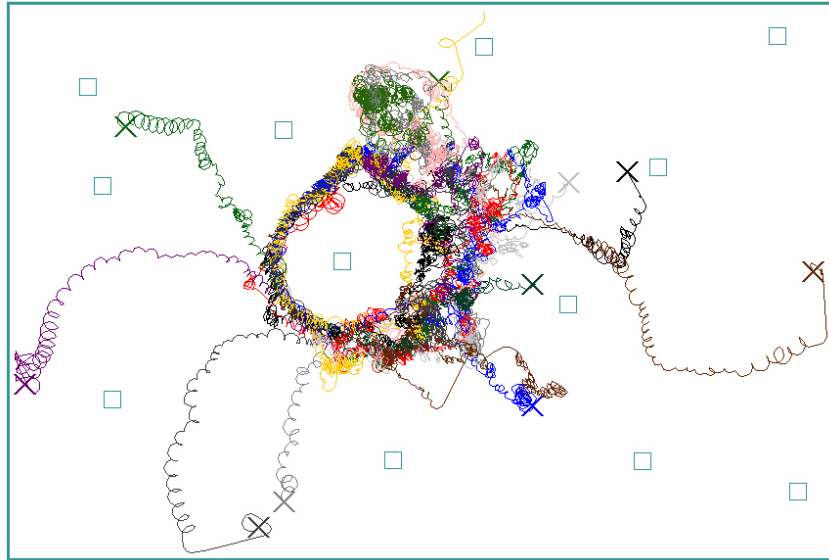


Abbildung 5.5: Trajektorien eines typischen Laufes des in Kapitel 5.2.2 beschriebenen Automaten.

sendet. Gleichzeitig aktiviert jedes Individuum regelmäßig seinen eigenen Broadcast, damit es auch von anderen wahrgenommen werden kann. Als Resultat suchen die Individuen die Nähe anderer, vermeiden die Bereiche um die in der Umgebung platzierten „Dummy-Roboter“, wie mittig in Abbildung 5.5 zu sehen ist, und bleiben gleichzeitig in Bewegung. Dieses Verhalten erfüllt, genau wie das zuvor entwickelte „Wall Avoid“ Verhalten, alle Aspekte der Bewertungsfunktion und wird somit als erfolgreich gewertet.

Der Umstand, dass dieses Verhalten bei der Broadcast-Dauer von einem Zyklus entwickelt wurde, lässt die Vermutung zu, dass die Broadcast-Dauer keinen wesentlichen Einfluss auf die Güte des entwickelten Verhaltens hat. Zwar wurde in den Simulationen mit einer Broadcast-Dauer von zwei und fünf Zyklen keine solche Verhaltensweise festgestellt, jedoch liegt der Prozentsatz der Simulationen, bei denen das „Wall-Avoid“ Verhalten entwickelt wird bei ca. 1%. Somit müsste die Anzahl der Simulationen wesentlich erhöht werden, um quantitative Aussagen treffen zu können. Zudem zeigt der Automat, wenn er in einer Umgebung mit einer verlängerten Broadcast-Dauer getestet wird, ein unverändertes Bewegungsmuster. Auch haben die Versuche gezeigt, dass das resultierende Verhalten invariant zu einer Erhöhung des Broadcast-Radius ist.

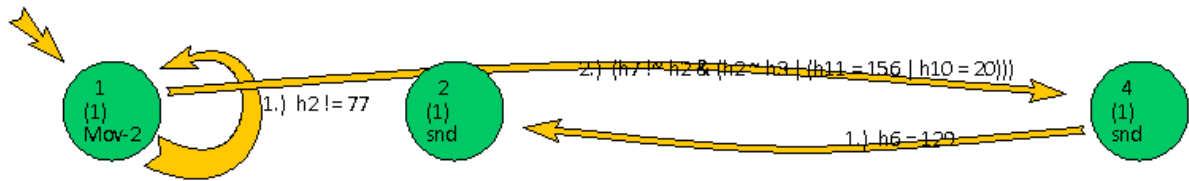


Abbildung 5.6: Typischer Automat, der zu einer umherfahrenden Verhaltensweise führt.

## 5.3 Ergebnisse „Futtersuche Szenario“

In diesem Kapitel werden nun die Ergebnisse der durchgeführten Simulationen im „Futtersuche Szenario“ vorgestellt. Bei der Analyse der einzelnen Simulationsläufe wurde schnell deutlich, dass sich die verschiedenen evolvierten Verhaltensweisen in Gruppen einteilen lassen, da sie sich im gezeigten Verhalten stark ähneln. Der Fall, dass sich kein Verhalten ausgebildet hat, also die Roboter stehen bleiben, ist mit der Futterplatzierung via „SetFutter“ nur selten vorgekommen. Die verschiedenen Gruppen werden im Folgenden anhand repräsentativer Beispiele vorgestellt.

### 5.3.1 Verhaltensweise 1: Fahren

Die am häufigsten vorzufindende Verhaltensweise ist das Umherfahren der Roboter. Ein typischer Automat, welcher zu dieser Verhaltensweise führt, ist in Abbildung 5.6 dargestellt. Der wichtigste Zustand des Automaten ist der Startzustand „Mov“, welcher den Roboter geradeaus fahren lässt. Richtungswechsel werden nicht durch den Automaten realisiert, sondern es wird eine Eigenschaft der Simulationsumgebung ausgenutzt. Diese dreht die Roboter bei Kollisionen mit anderen oder mit Wänden um einen zufälligen Winkel, was in frühen Zyklen der Simulation ein besseres Anlaufen der Evolution ermöglichen soll. Auf diese Weise fahren die Roboter zufällig durch die Umgebung und treffen in seltenen Fällen auf eine Futterstelle. So verbessert sich kurzfristig ihre Güte, was zu den positiven Ausschlägen der Gütefunktion in Abbildung 5.6 führt. Da jede Futterquelle ca. 1,29% der Umgebung bedeckt und mit einer maximalen Futterquellenanzahl von 5 getestet wurde, die Umgebung also zu über 93% futterfrei ist, befinden sich die Roboter häufiger außerhalb einer Futterstelle als innerhalb. Somit überwiegt der Futtermalus gegenüber dem Bonus, sodass die durchschnittliche Güte negativ ist. Der hier vorgestellte Automat stellt einen Spezialfall dar. So beinhaltet er auch „Snd“ Zustände, welche einen

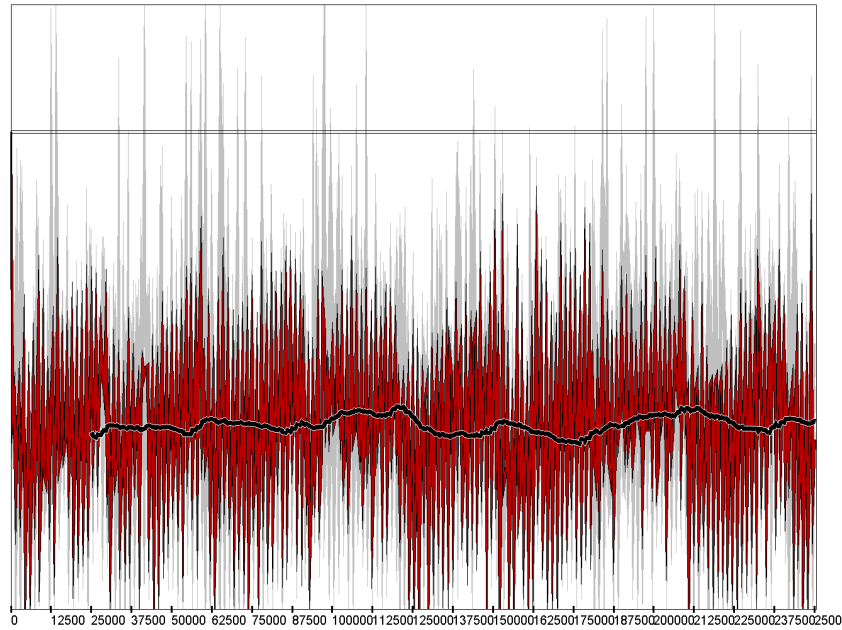


Abbildung 5.7: Verlauf der Güte, welche während eines typischen Laufes mit umherfahrenden Robotern aufgenommen wurde. Auf der X-Achse sind die Simulationszyklen, auf der Y-Achse die Güte abgetragen. Die grauen Linien zeigen den Verlauf der Güte der einzelnen Individuen, die roten Linie die durchschnittliche Güte der Population zu einem bestimmten Zeitpunkt. Der schwarze Graph stellt den gleitenden Durchschnitt der durchschnittlichen Güte über 50000 Zyklen dar.

Broadcast starten. Somit ist Kommunikation, wie in Kapitel 2 definiert, in diesem Szenario vorhanden, da die Individuen Signale austauschen. Allerdings sind keine Anzeichen für Sprache vorhanden. Hierzu müssten zum einen Informationen an den Broadcast geknüpft sein, zum anderen müssten diese Informationen vom Empfänger interpretiert und verwendet werden. Während der erste Schritt zur Sprache, das Knüpfen von Informationen an den Broadcast, durchaus durch die Zustandsübergangsbedingungen realisiert sein kann, werden die Informationen vom Empfänger nicht zur Verbesserung der Güte verwendet. Zudem ist die Bedingung des Zustandsüberganges von „Mov“ zu „Snd“ nur selten erfüllt, weshalb die Roboter die meiste Zeit schweigen. Deshalb wurden Simulationen, in denen die Roboter die umherfahrende Verhaltensweise zeigten, nicht als erfolgreich gewertet.



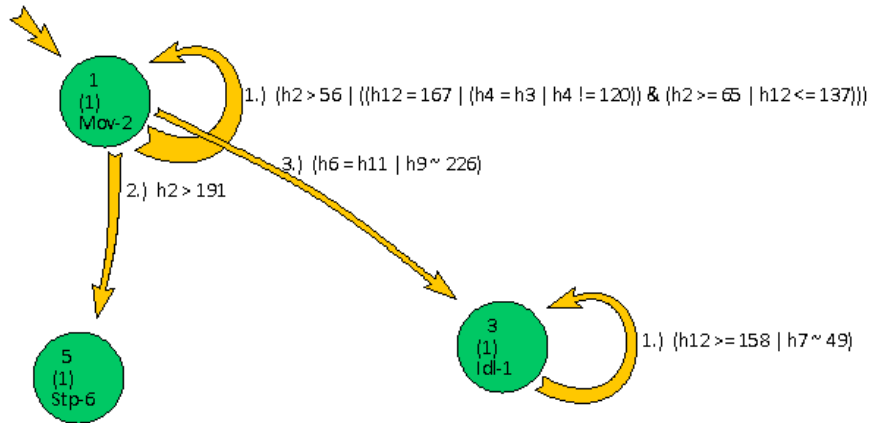


Abbildung 5.8: Automat, welcher ein stoppendes Verhalten zeigt.

### 5.3.2 Verhaltensweise 2: Stoppen

Während der Simulationen haben sich verschiedene Verhaltensweisen entwickelt, bei denen die Roboter stoppen, sobald sie sich in der Nähe einer Futterquelle befinden. Im Folgenden wird diese Verhaltensgruppe exemplarisch an dem in Abbildung 5.8 gezeigten Automaten erläutert. Dieser lässt die Individuen der Population zunächst zufällig die Umgebung erkunden. Da die Bedingungen  $h4 \neq 120$  und  $h12 \leq 137$  meist erfüllt sind, verweilt der Automat solange im „Mov“ Zustand, bis er eine Futterquelle erreicht. Danach wechselt er in Zustand 3, dem „Idl“ Zustand. Dieser lässt den Roboter stoppen. Ist hier die Bedingung  $h12 \geq 158$  erfüllt, befindet sich der Roboter also weit genug innerhalb der Futterquelle, verlässt er diesen Zustand solange nicht, bis die Futterquelle geleert ist. Das von diesem Automaten entwickelte Verhalten ist zwar relativ unspektakulär, doch wird es vielen Aspekten der Gütefunktion gerecht und ist gerade durch seine Einfachheit sehr robust.

### 5.3.3 Verhaltensweise 3: Drehen

Eine weitere evolvierte Verhaltensweise ist das Drehen der Roboter. Diese kennzeichnet sich dadurch, dass die Roboter kreisende Bewegungen ausführen, welche mehr oder weniger stark ausgeprägt sein können.

Der Automat, welcher die schwächste Ausprägung der drehenden Verhaltensweise zeigt,

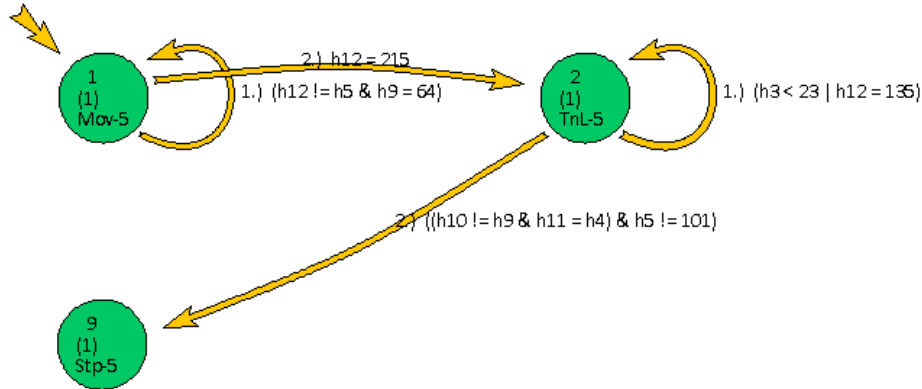


Abbildung 5.9: Automat, welcher schwache Ausprägung des drehenden Verhaltens zeigt.

ist in Abbildung 5.9 zu sehen. Die von diesem Automaten kontrollierten Roboter fahren zunächst zufällig in der Umgebung umher, bis sie auf eine Futterstelle stoßen. Wenn die Bedingung  $h12 = 215$  erfüllt ist, der Roboter sich also relativ mittig auf der Futterquelle befindet, wechselt er in den „TnL“ Zustand, welcher eine Linksdrehung des Roboters bewirkt. Falls die Bedingung  $h3 < 23 \vee h12 = 135$  erfüllt ist, verweilt der Roboter in diesem Zustand, weshalb er sich dauerhaft um die eigene Achse dreht und so innerhalb der Futterquelle bleibt. Betrachtet man diese Bedingung genauer, fällt auf, dass der rechte Teil der Disjunktion nicht erfüllt sein kann, da der Automat nur in den „TnL“ Zustand gelangt, wenn  $h12 = 215$  gilt. Folglich muss  $h3 < 23$  gelten, damit der Roboter die Futterquelle nicht einfach nur durchfährt. Diese Bedingung ist immer dann erfüllt, wenn sich weder eine Wand noch ein anderer Roboter rechts des betrachteten befindet.

Die Schwächen dieser Verhaltensweise offenbaren sich direkt. Zum Ersten ist die Bedingung  $h12 = 215$  wenig robust. So kann beispielsweise durch Rundungen der Sensorwert 215 übersprungen werden oder der Roboter durchquert die Futterquelle nicht mittig genug, sodass die angenommenen Sensorwerte kleiner als 215 sind. Zum Zweiten darf sich keine Wand und kein anderer Roboter in der Nähe des Roboters befinden. Dies führt dazu, dass sich nur selten mehrere Roboter an derselben Futterquelle befinden. Auf diese Weise werden die Roboter nie die volle Anzahl Boni erhalten. Die dritte Schwäche des Verhaltens offenbart sich, sobald die Futterquelle geleert und an einer neuen Position wieder initialisiert wurde. Da die Bedingung  $h3 < 23$  unabhängig von einer Positionierung auf einer Futterquelle ist, wird der Roboter sich auch nach dem Verschwinden der Futterquelle weiter im Kreis drehen und sich aus eigener Kraft nicht aus dieser Endlosschleife

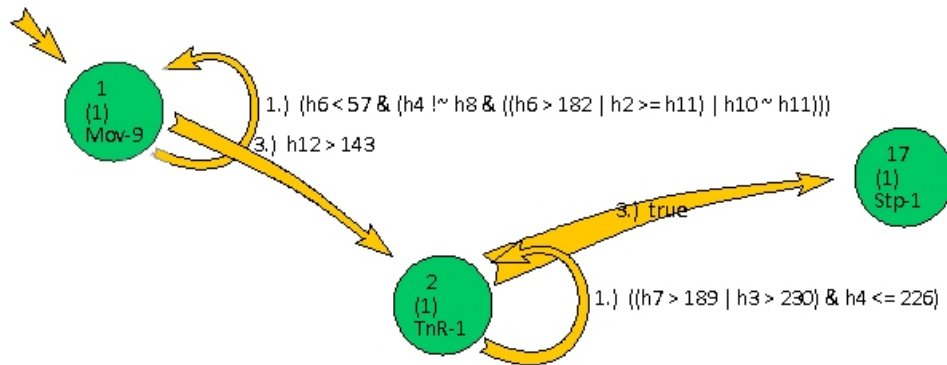


Abbildung 5.10: Automat, welcher die zweite vorgestellte Ausprägung des drehenden Verhaltens zeigt.

befreien können.

Eine robustere drehende Verhaltensweise zeigt der in Abbildung 5.10 dargestellte Automat. Dieser lässt die Individuen, ebenso wie der zuvor beschriebene Automat, zunächst zufällig in der Umgebung umherfahren. Erreicht ein von diesem Automat gesteuerter Roboter eine Futterquelle, so ist die Übergangsbedingung 3:  $h12 > 143$  erfüllt, falls der Roboter die Futterquelle mittig genug durchfährt. Die Fläche der Futterquelle, in der diese Bedingung erfüllt ist, ist wesentlich größer als im vorherigen Fall. Somit ist es wahrscheinlicher, dass der Roboter seinen aktuellen Zustand wechselt. Wird der Zustandswechsel vollzogen, dreht sich der Roboter nach rechts, bevor er für einen Zyklus stoppt. Dieses Bewegungsmuster - Fahren - Drehen - Stoppen - wiederholt er so lange, bis die Bedingung  $h12 > 143$  nicht mehr erfüllt ist, die Futterquelle also entweder versiegt ist oder er sich wieder außerhalb der Futterquelle befindet. Hier liegt ein weiterer Vorteil im Vergleich zum vorherigen Automaten, da dieser nach dem Verschwinden der Futterquelle den Roboter eigenständig mit der Suche nach einer neuen Futterstelle beginnen lässt. Doch auch dieser Automat zeigt noch Schwächen. So kann es passieren, dass sich der Roboter durch die vollzogene Kreisbewegung weiter zum Rand der Futterstelle dreht und sich danach von dieser entfernt, da der Futtersensorwert kleiner als 143 ist. Zwar verlängert er in diesem Fall durch sein Stoppen die Verweildauer innerhalb der Futterquelle um ca. 33%, doch ist auch dieses Verhalten durchaus noch verbesserungswürdig.

Die dritte und robusteste evolvierte Art eines drehenden Verhaltens unterscheidet sich entscheidend von den zwei bisher vorgestellten. Zwar ist der Startzustand auch ein „Mov“ Zustand (vgl. Abb 5.11), das resultierende Verhalten kennzeichnet sich jedoch durch ei-

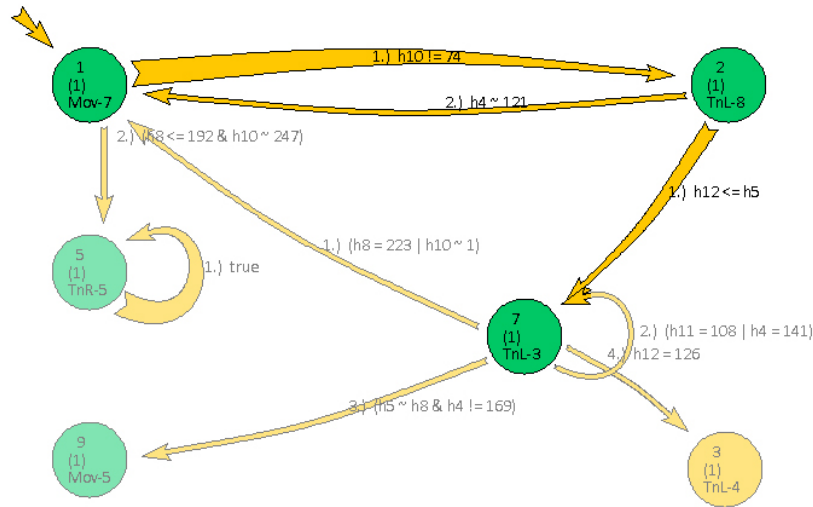


Abbildung 5.11: Automat, welcher die beste vorgestellte Ausprägung des drehenden Verhaltens zeigt.

ne konstante Drehbewegung, ähnlich wie die in Kapitel 7.1.1 vorgestellten, in früheren Simulationen entstandenen Verhaltensweisen. Die Übergangsbedingung von Zustand 1, dem „Mov“ Zustand, zum zweiten Zustand, welcher für eine Linksdrehung sorgt, lautet  $h10 \neq 74$ . Da keiner der Automaten innerhalb der Population einen „Snd“ Zustand enthält, ist diese Bedingung immer erfüllt. Der Zustand, welcher dafür sorgt, dass die Roboter keine regelmäßigen Kreise fahren, ist der 7. Zustand, welcher ebenfalls ein „TrL“ Zustand ist. Dieser kann vom zweiten Zustand aus erreicht werden, wenn die Bedingung  $h12 \leq h5$  erfüllt ist, sich also keine Futterquelle vor dem Roboter befindet. Die Trajektorien eines beispielhaften Laufes mit Robotern, welche vom zuletzt vorgestellten Automaten gesteuert werden, ist in Abbildung 5.12 dargestellt. Wie zu erkennen ist, drehen sich Roboter ohne eine Futterquelle in ihrer Nähe in kleinen Kreisen. Roboter, in deren Sensorreichweite sich eine Futterquelle befindet, drehen sich nur dann in engen Kreisen, Durchlaufen also die Zustände Fahren - Drehen - Drehen, wenn sie das Futter nicht sehen können. Sobald die Futterquelle durch die Drehbewegung wieder in ihren Sensorbereich rückt, ändert sich die Zustandsabfolge zu Fahren - Drehen. Somit vergrößert sich kurzzeitig der Radius des gefahrenen Kreises, wodurch sich die Roboter langsam der Futterquelle annähern. Die aus diesem Automaten resultierende Verhaltensweise hat den Vorteil gegenüber den anderen hier vorgestellten, dass sie, falls eine Futterquelle in der Sensorreichweite liegt, diese auch findet und darauf zusteuert. Ist dies allerdings nicht der Fall, erkunden die Roboter die Umgebung nicht eigenständig, wie in den beiden vorherigen Fällen. Zudem ist ihre Fortbe-

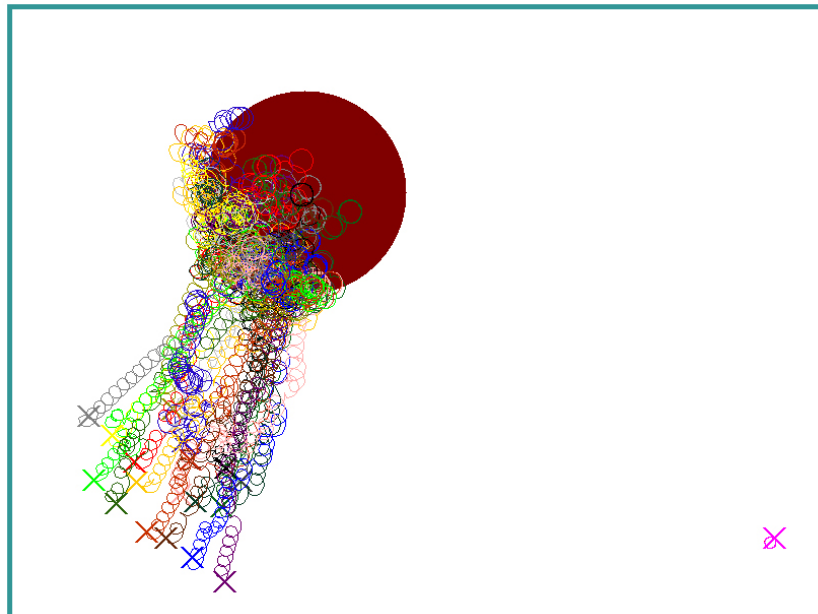


Abbildung 5.12: Trajektorien, welche zeigen, wie sich der in Abbildung 5.11 gezeigte Automat einer Futterquelle annähert.

wegung durch die konstanten Drehbewegungen sehr langsam. Trotzdem erzielt die letzte drehende Verhaltensweise bessere Ergebnisse hinsichtlich der Gütefunktion bei gleicher Anzahl an Futterquellen in der Umgebung.

#### 5.3.4 Verhaltensweise 4: Stehen

In anfänglichen Simulationen, in denen die Futterquellen nicht mithilfe des „SetFutter“ Algorithmus platziert wurden, konnte häufig beobachtet werden, dass die Population keine Fortbewegung entwickelte. Diese Problematik konnte mit der Einführung des Algorithmus aufgehoben werden, jedoch entwickeln sich vor allem in Simulationen mit langen Futterverweilzeiten Verhaltensweisen, bei denen die Individuen stehen bleiben ohne jedoch eine stoppende Verhaltensweise evolviert zu haben. Diese Simulationen kennzeichnen sich durch einen zu Abbildung 5.13 ähnlichen Güteverlauf. Stehen viele Individuen auf einer Futterstelle, steigt die Güte der Population kurzzeitig. Sobald die Futterquelle geleert ist, verweilen die Individuen jedoch an ihrer Position. Somit sinkt ihre Güte und bleibt für längere Zeit auf einem niedrigen Niveau konstant.

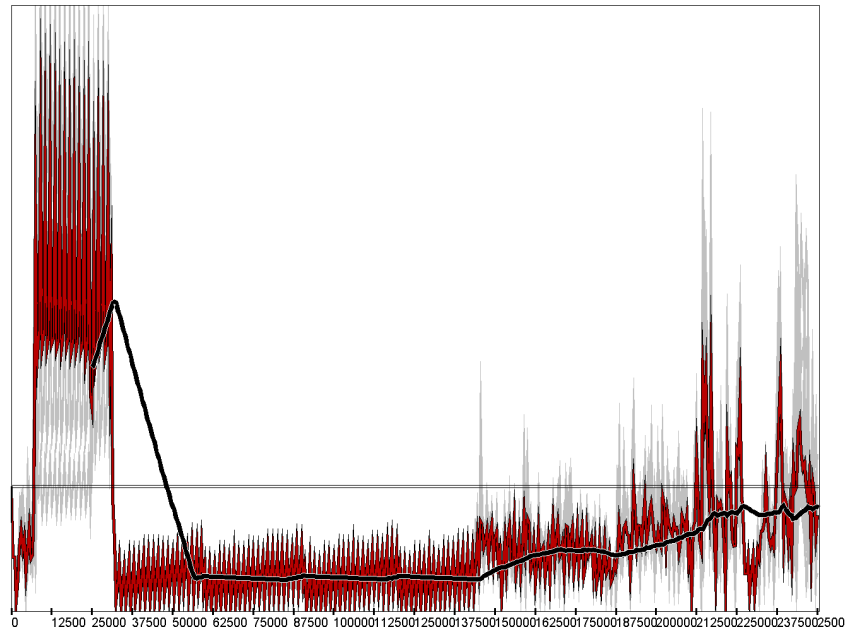


Abbildung 5.13: Güteverlauf eines typischen Laufes, bei welchem Individuen durch Stehen kurzzeitig viel Güte erhalten

### 5.3.5 Zusammenfassung der Ergebnisse

Alle hier vorgestellten Verhaltensweisen weisen keine Anzeichen von Sprache auf. Das bedeutet jedoch keinesfalls, dass die Evolution von Sprache in diesem Szenario nicht möglich ist. Vielmehr zeigt dies, dass die gewählte Aufgabenstellung, wie beabsichtigt, auch ohne den Einsatz von Sprache lösbar ist, allerdings potenziell nicht so gut. Zudem, so Lipson [Lip07, S. 331], sei die Evolution von Sprache deshalb so schwierig, da sich Sender und Empfänger parallel zueinander entwickeln müssen. Um den Anreiz für die Evolution von Sprache in den Untersuchungen zu vergrößern, stehen verschiedene Parameter zur Verfügung, deren Einfluss im Folgenden analysiert wird.

Ein potenzieller Grund, warum keine Sprache evolviert wird, ist, dass wenn die Individuen gezielt zu den Futterstellen navigieren auch ohne Kommunikation mehrere Roboter an einer Futterstelle vorzufinden sind, da alle Roboter im Sensorumkreis auf diese zusteuern. Um diesen Effekt zu umgehen, bestehen verschiedene Möglichkeiten. Zum einen wird mit einer Belohnung experimentiert, welche proportional zur Anzahl der an einer Futterquelle befindlichen Individuen ist und somit immer der Anreiz besteht, mehr Roboter zur eigenen Futterquelle zu locken. Dies hat aber den Nebeneffekt, dass die Roboter häufiger ein rein

drehendes Verhalten zeigen, bei dem sie nicht aktiv nach Futterquellen suchen. Vielmehr verharren sie in den Großgruppen, welche sich mit der Zeit bilden, wenn sich jeder in der Umgebung einer Futterstelle befindliche Roboter auf diese zubewegt. Der Bonus, welchen sie für andere Individuen an derselben Futterquelle erhalten, wird dann so groß, dass er den Malus, welchen die Roboter für die Zeit erhalten, in welcher sie sich in dem futterlosen Teil der Umgebung im Kreis drehen, zu überwiegen scheint. Eine andere Möglichkeit die Roboter zum Sprechen zu animieren ist die Anzahl der Futterquellen zu erhöhen, damit sich potenziell weniger Individuen an derselben Futterstelle befinden. Dies hemmt jedoch anscheinend die Entwicklung komplexer Verhaltensweisen, weshalb der Anteil an umherfahrenden Robotern in Simulationen mit fünf Futterquellen höher ist als in Simulationen mit ein oder zwei Futterquellen. Theoretisch ist es auch möglich, dass während einer Simulation Sprache entwickelt wird, welche den anderen Robotern innerhalb der Umgebung die Position einer Futterquelle signalisiert. Da andere Roboter jedoch Zeit benötigen die Futterquelle zu erreichen, könnte es passieren, dass diese in der Zwischenzeit versiegt. Um diesen Effekt zu verhindern, wird ebenfalls mit einer erhöhten Futtermenge experimentiert. Allerdings zeigt sich in den Simulationen, dass eine zunehmende Futtermenge einen negativen Einfluss auf die Evolution zu Beginn einer Simulation hat. So entwickeln Individuen, welche in einer Umgebung mit hoher Futtermenge pro Futterplatz evolvieren, wesentlich später erkennbares Verhalten. Dies lässt sich beispielsweise dadurch erklären, dass ein Roboter, während er über eine Futterstelle fährt, so mutiert, dass er stehen bleibt. Da er so eine sehr viel höhere Güte erreichen würde, als die restliche Population, würde er seinen Automaten verstärkt weitergeben. Dies würde dazu führen, dass die gesamte Population allmählich stehen bliebe, ohne jedoch ein Verhalten erlernt zu haben.

## 5.4 Platzieren der Futterplätze

Um zu evaluieren, wie gut der in Kapitel 4.4.1 vorgestellte Algorithmus 2 arbeitet, wurde ein neues Plug-In geschrieben. Dieses testet den Algorithmus 2, nachfolgend „SetFutter“ genannt, gegen einen Algorithmus, welcher x- und y-Koordinate der Futterstelle unabhängig und rein zufällig generiert, im Weiteren als zufälliger Algorithmus bezeichnet. Maßgebend für die Güte der Algorithmen ist nicht deren Laufzeit, sondern ihre Fehleranzahl. Ein Fehler liegt immer dann vor, wenn eine Futterquelle nicht überschneidungsfrei platziert werden kann, also auf einem Roboter oder einer anderen Futterstelle platziert

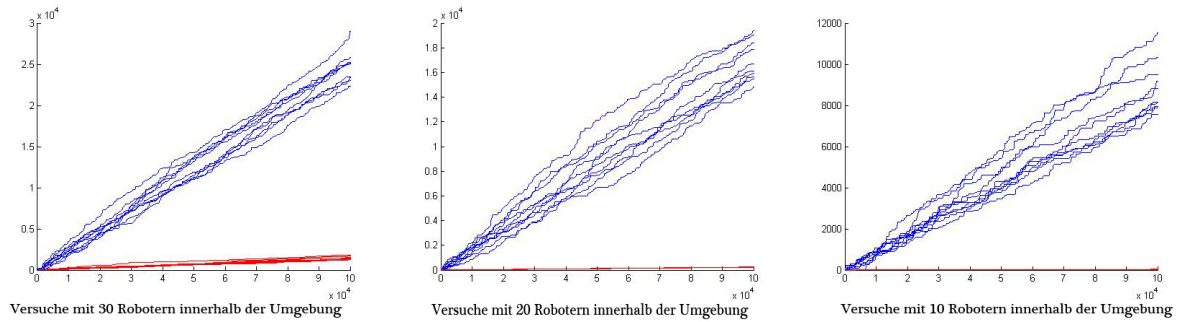


Abbildung 5.14: Vergleich der Anzahl der nicht überschneidungsfreien Platzierungen von Futterplätzen bei rein zufälliger Platzierung (blau) oder Platzierung durch den Algorithmus 2 (rot)

wird. Da der zum Setzen der Futterquellen verwendete Algorithmus „SetFutter“ nie eine Futterquelle platziert für die kein Platz existiert, wird bei ihm das Erreichen des Abbruchkriteriums als Fehler angesehen. Dieses garantiert das Terminieren des Algorithmus und ist dann erreicht, wenn er nach 300 Durchläufen keinen Platz gefunden hat, an dem er die Futterquelle setzen kann. Um Chancengleichheit zu sichern, denn der „SetFutter“ Algorithmus durchläuft 300 Iterationsschritte bevor er einen Fehler begeht, wird der zufällige Algorithmus erweitert. Vor jedem Versuch eine Futterquelle zu setzen, wird auf Überschneidungsfreiheit getestet. Falls diese nicht gegeben ist, wird eine neue Position ermittelt. Dies wird solange wiederholt, bis entweder eine Futterstelle überschneidungsfrei gesetzt werden kann oder der Vorgang des zufälligen Ziehens und anschließender Überprüfung 300 mal durchlaufen wurde. Beide Algorithmen werden in Simulationen gegeneinander getestet. Hierzu werden zehn, zwanzig und dreißig Roboter in einer leeren Umgebung verteilt. Alle Roboter enthalten dabei Automaten, welche lediglich aus einem „Mov“-Zustand bestehen, sodass sie zufällig in der Umgebung umherfahren, da sie nach der Grunddynamik bei Kollisionen untereinander oder mit Wänden um einen zufälligen Winkel gedreht werden. Somit erkunden die Roboter zufällig die gesamte Umgebung. Beide Algorithmen haben in jedem Zyklus die Aufgabe eine Futterquelle zu platzieren. Kollidieren diese mit Robotern, erhalten die Algorithmen einen Fehlerpunkt, welcher gespeichert und schließlich in eine Textdatei übertragen wird. Die Futterquellen wurden allerdings nicht wirklich gesetzt, da ansonsten bei einer Laufzeit von 100000 Zyklen die Umgebung aufgrund von Überfüllung schnell kein überschneidungsfreies Platzieren mehr ermöglichen würde. Diese Versuche werden für die verschiedenen Anzahlen von Robotern



Tabelle 5.2: Kumulierte Fehler beim Setzen der Futterquellen

Versuch	10 Roboter		20 Roboter		30 Roboter	
	SetFutter	Zufall	SetFutter	Zufall	SetFutter	Zufall
1	34	10318	233	16124	1351	25251
2	34	7887	229	15931	1832	25008
3	37	9160	245	18371	1302	25828
4	35	7557	234	15451	1234	23372
5	35	11532	228	14851	1684	25076
6	35	9494	222	15641	1338	23059
7	36	8810	232	17857	1370	25114
8	35	8158	228	19026	1486	28951
9	34	8172	233	16707	1295	22338
10	35	7937	236	19411	1505	23454
Mittelwerte	35	8902,5	232	16937	1439,7	24745,1

jeweils 10mal ausgeführt, sodass in der Summe 3 Millionen Testdaten erhoben werden. Diese Summenfunktionen der einzelnen Fehler sind in Abbildung 5.14 dargestellt.

Wie deutlich zu sehen ist, ist die Fehlerrate des „SetFutter“ Algorithmus deutlich niedriger als bei zufälliger Platzierung der Futterquellen. So liegt die durchschnittliche Fehlerquote des zufälligen Algorithmus in einer Umgebung mit 30 Robotern bei ca. 25%, während der „SetFutter“ Algorithmus eine maximale Fehlerquote von 1,8%, im Durchschnitt sogar nur von ca. 1,5% aufweist. Dies verdeutlicht die Notwendigkeit, einen eigenen Algorithmus zu implementieren. Würde jede vierte Futterquelle unterhalb eines Roboters initiiert, würde in der Evolution keine Notwendigkeit für die Roboter bestehen sich zu bewegen oder ein Verhalten zu evolvieren.



## 6 Zusammenfassung

In dieser Arbeit wurde in zwei verschiedenen Szenarien getestet, ob sich Sprache in Roboterschwärmen, welche von Endlichen Automaten gesteuert werden, entwickelt. Im ersten Szenario, dem „Sozialen Szenario“, wurde den Robotern die Aufgabe gestellt, mit Hilfe von Sprache andere Roboter von Wänden zu unterscheiden. Auch die weiterführenden Untersuchungen dieser Arbeit konnten belegen, dass das in bisherigen Untersuchungen zu diesem Thema [WK10] vorgestellte „Wall Avoid“ Verhalten nicht ohne Sprache zu erklären ist.

Im zweiten untersuchten Szenario, dem „Futtersuche Szenario“, bestand die Aufgabe der Roboter darin, sich zusammen mit anderen auf Futterstellen aufzuhalten, welche in der Umgebung verteilt wurden. In Simulationen wurde gezeigt, dass die Individuen auf verschiedene Weisen diese Aufgabe erfüllen, ohne jedoch Sprache zu nutzen. Dies spiegelt die Schwierigkeit des Ansatzes wieder. Dieser war genau so gewählt worden, dass anders als im „Sozialen Szenario“ Sprache nicht zwingend erforderlich, jedoch in Hinblick auf die Güte der Individuen förderlich ist. Durch Sprache sollten anderen Individuen, welche aufgrund der beschränkten Futersensorreichweite keine Futterquelle wahrnehmen konnten, die Position der eigenen Futterquelle mitgeteilt werden.

### 6.1 Ausblick

Diese Arbeit schöpft bei weitem nicht alle Möglichkeiten der vorgestellten Szenarien aus. So könnten in zukünftigen Untersuchungen speziell für das „Futtersuche Szenario“ noch weitere Parametertests durchgeführt werden, mit dem Ziel die Evolution von Sprache zu forcieren. Beispielsweise könnte hierzu der Futersensorradius weiter verringert werden oder die Belohnung für das alleinige Aufhalten auf einer Futterquelle entfallen. Zudem könnte beispielsweise durch Variieren der Anzahl der in der Reproduktion der Individuen

verwandten Eltern die Diversität innerhalb der Population vergrößert werden und so die gleichzeitige Evolution von Sender und Empfänger erleichtern. Falls Sprache evolviert wird, könnte die Aufgabenstellung auf das reale Problem des fairen Aufladens der Roboter übertragen werden oder die Simulation um eine Physik-Engine erweitert werden, welche den physischen Abtransport des Futters simuliert.





# 7 Anhang

## 7.1 Veraltete Verhaltensweisen

In diesem Kapitel werden Verhaltensweisen beschrieben, welche auf fehlerhafte Implementierungen aufbauten.

### 7.1.1 Verhaltensweise: Drehen

Ein Teil der Simulationsläufe wurde mit einer fehlerhaften Futtersensorik durchgeführt, hervorgerufen durch einen Fehler in der Schnittpunktberechnung der Kanten des Sensorbereichs mit den Futterquellen. So nahm der Futtersensorwert in manchen Fällen, wenn der Roboter parallel zur Futterstelle fuhr, fälschlicherweise den Wert 0 an. Dies ermöglichte das Drehen der Roboter. Diese Verhaltensweise kennzeichnet sich durch ein höheres aber immer noch negatives Niveau an durchschnittlicher Güte, verglichen mit der Güte des reinen Umherfahrens, wie in Kapitel 5.3.1 beschrieben. Außerdem zeigt sich meist

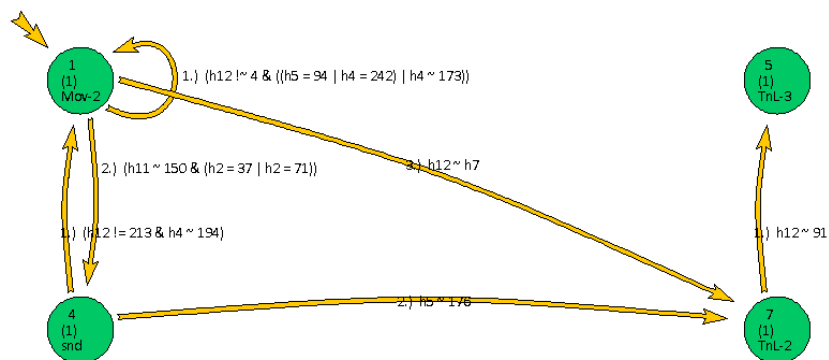


Abbildung 7.1: Automat, welcher eine drehende Verhaltensweise zeigt

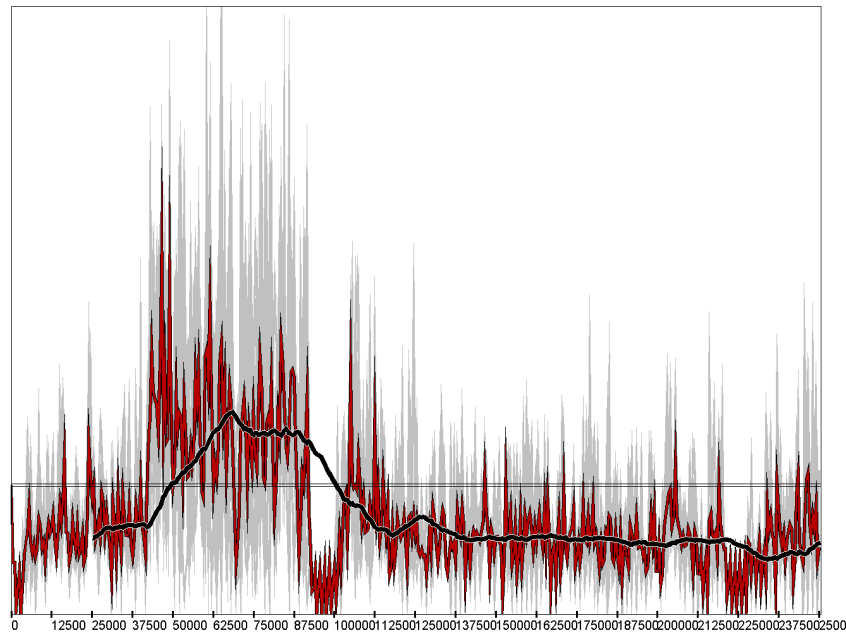


Abbildung 7.2: Gütefunktionsverlauf eines typischen Laufes mit Individuen, welche eine drehende Verhaltensweise zeigen

ein sprunghafter Anstieg des gleitenden Durchschnitts der Gütefunktion, manchmal sogar in den positiven Wertebereich, wie in Abbildung 7.2 zu sehen. Ein typischer Automat, welcher ein Drehen der Individuen verursacht, ist in Abbildung 7.1 abgebildet. Allen „Drehern“ ist gemein, dass ihr Anfangszustand ein „Mov“ Zustand ist. Von diesem geht eine Kante zu einem „Tnl“ oder „Tnr“ Zustand, je nachdem, ob sie sich gegen den oder im Uhrzeigersinn drehen. Die Übergangsbedingung der Kante beinhaltet im Wesentlichen die Bedingung „ $h12 \leq x$ “, wobei  $h12$  den Futtersensor-Sensorwert repräsentiert und  $x$  eine meist auf 0 gesetzte Schwelle ist. Das aus diesem Automat resultierende Verhalten lässt sich wie folgt beschreiben. Falls sich keine Futterquelle im Radius des Futtersensors um einen Roboter befindet, dreht sich dieser im Kreis, da die Bedingung „ $h12 \leq 0$ “ erfüllt ist. Sobald eine Futterquelle in der Nähe des Roboters platziert wird, sodass der Futtersensor einen Wert ungleich 0 anzeigt, fährt der Roboter auf sie zu. Falls er nicht genau auf die Futterquelle zusteuert, wird der Sensorwert nach einer gewissen Zeit wieder den Wert 0 annehmen, da der Sensorbereich seitlich an der Futterquelle vorbei schaut. Ist dies der Fall, dreht sich der Roboter wieder so lange um die eigene Achse, bis die Futterquelle wieder innerhalb seines Sensorbereiches liegt. Auf diese Art und Weise gelingt es ihm, die Futterquelle zu erreichen.



Da alle Individuen der Population, welche sich im Umkreis einer Futterquelle befinden, auf diese zufahren, bilden sich schnell Gruppen von Robotern, sodass diese auch den Gruppenbonus der Gütefunktion erhalten, obwohl keine Kommunikation vorhanden ist. So ist das erhöhte Güteniveau im Vergleich zu nur fahrenden Robotern zu erklären. Sobald die Futterquelle geleert ist, drehen sich die Roboter wieder im Kreis und warten darauf, dass eine neue Futterquelle in ihrer Nähe platziert wird. Dies ist nicht zu vermeiden, da sich mit der Zeit verschiedene Kleingruppen von drei bis fünf Individuen in der Umgebung verteilen und so fast die gesamte Umgebung im Sensorbereich der Population liegt.

Ein Sprung in der durchschnittlichen Güte ist dann festzustellen, wenn zwei oder mehr Futterquellen nacheinander dicht bei derselben Robotergruppe platziert wurden. Die Roboter leeren dann erst eine Futterquelle, bevor sie zur nächsten wechseln. Auf diese Weise erhalten sie in kurzer Zeit eine hohe Güte, da das von ihnen gezeigte Verhalten allen Aspekten der Gütefunktion entspricht. Sobald allerdings keine Futterstelle mehr in ihrer Nähe ist, fällt ihre Güte wieder auf das Maß der restlichen Population zurück. Mit einer anderen Parametrisierung, in der die Roboter mit ihrem Futtersensor die gesamte Umgebung überblicken können, ist die drehende Verhaltensweise sogar nahezu optimal. Da die Roboter immer eine Futterquelle finden, befinden Sie sich dauerhaft auf dem Weg zu dieser. Die Zeit, in der sie wartend im Kreis fahren, entfällt.

Manche Roboter haben darüber hinaus eine weitere Verfeinerung des drehenden Verhaltens entwickelt. Bei ihnen ist der Schwellenwert  $x$  des Zustandsüberganges nicht fest, sondern von einem anderen Sensor abhängig. Bei rechtsdrehenden Robotern lautet die Bedingung beispielsweise „ $h_{12} \leq h_7$ “, wobei  $h_7$  der vordere linke Infrarotsensor ist. So versuchen Roboter Hindernissen, welche sich links vor ihnen befinden, durch eine Rechtsdrehung auszuweichen.

Mit korrekter Sensorik verändert sich das Verhalten dahingehend, dass die Roboter immer genau so weit drehen, bis die Kante der Futterstelle in ihren Sensorbereich dringt, bevor sie wieder gradeaus fahren. Mit falscher Sensorik ist dies erst später der Fall, sodass sich die Roboter so weit drehen, dass sie wieder zurück in die Futterquelle fahren (vgl. Abb. 7.4). Wird der Sensorwert korrekt berechnet, fahren sie schon so früh wieder gradeaus, dass sie an der Futterquelle vorbeifahren. Auf diese Weise kreisen sie lediglich um die Futterquelle (vgl. Abb. 7.3) ohne auf sie zu gelangen. Die einzige Möglichkeit die Futterquelle zu durchqueren ist, mit anderen Robotern zu kollidieren und so die Endlosschleife kurzzeitig zu durchbrechen. Sobald die Durchquerung der Futterquelle abgeschlossen ist, fahren die

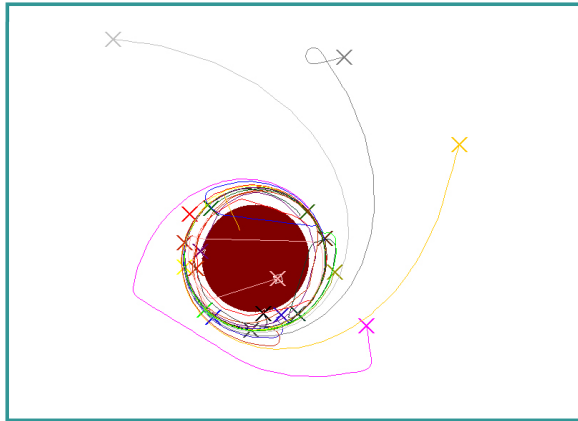


Abbildung 7.3

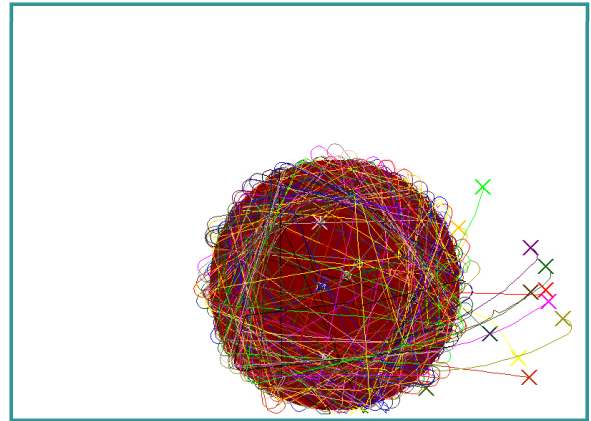


Abbildung 7.4

Trajektorien der drehenden Verhaltensweise bei korrekter Sensorik (Abb. 7.3) sowie bei fehlerhafter Sensorik (Abb. 7.4)

Roboter mit ihrer kreisenden Bewegung fort. Somit ist die Verhaltensweise im Fall der fehlerhaften Sensorik relativ erfolgreich, da sie viele Aspekte der Gütefunktion erfüllt, im Fall der korrekten Sensorik allerdings vollkommen nutzlos.

Dieses Beispiel zeigt deutlich, wie sehr schon kleine Änderungen in der Sensorik das gesamte darauf aufbauende Verhalten beeinflussen können, da die Roboter mit ihrer Umgebung ein dynamisches System bilden.

## 7.2 Standardparametersatz

Tabelle 7.1: Standardparametersatz\*

Parameter	Wert	Bedeutung
explen	300000	Anzahl der Zyklen einer Simulation
umgebung	umgebung.bmp	Umgebung, in der die Roboter agieren, vgl. 4.5
verzerrung	0.3	Verzerrung der Simulation, Roboter haben 30% ihrer ursprünglichen Größe
evolution	true	Gibt an, ob evolutionäre Methoden wie Rekombination oder Mutation verwendet werden
fit	50	Anzahl der Zyklen, nach denen die Güte der Individuen berechnet wird
fitredwert	2.0	Faktor, durch den die Güte der Roboter geteilt wird, damit alte Verhaltensweisen mit zunehmender Zeit immer geringeren Einfluss auf die Güte des Individuum haben
fitredzyk	300	Anzahl der Zyklen, nach denen Robotergüte durch den Faktor „fitredwert“ geteilt werden
mut	300	Anzahl der Zyklen, nach denen mutiert wird
mutartverh	altemut	Verwandter Mutationsoperator, [KS08, Seite 88]
rek	300	Anzahl der Zyklen, nach denen rekombiniert wird
rekart	trivial	Verwandter Rekombinationsoperator**
rekeltern	6	Anzahl der Eltern der Rekombination
fitnessverfahren	sozial / futtersuche	Je nach Aufgabenstellung wird entweder die Soziale- oder Futtersuche-Gütefunktion genutzt

\* Lediglich für die Simulation relevante Parameter sind gelistet.

\*\* Im eigentlichen Sinne kein Rekombinations- sonder vielmehr ein Reproduktionsoperator, vgl. Kapitel 3.4.



# Literaturverzeichnis

- [Arm08] ARMSTRONG, David F.: The Gestural Theory of Language Origins. In: *Sign Language Studies* (2008), Nr. 3
- [Bal10] BALTER, Michael: Animal Communication Helps Reveal Roots of Language. In: *Science Magazine* 328 (2010), May, Nr. 5981, S. 969–971
- [Bar05] BARTELME, Norbert: *Geoinformatik: Modelle, Strukturen, Funktionen*. Berlin Heidelberg : Springer, 2005
- [BB06] BEHNKE, Joachim ; BEHNKE, Nathalie: *Grundlagen der statistischen Datenanalyse*. Wiesbaden : VS Verlag für Sozialwissenschaften, 2006
- [BL04] BUTTELMANN, Maik ; LOHMANN, Boris: Optimierung mit Genetischen Algorithmen und eine Anwendung zur Modellreduktion. In: *Automatisierungstechnik* (2004), April, Nr. 52, S. 151–163
- [Dar59] DARWIN, Charles: *On the Origin of Species – A Facsimile of the First Edition*. Cambridge : Harward University Press, 1859
- [FMMK07] FLOREANO, Dario ; MITRI, Sara ; MAGNENAT, Stéphane ; KELLER, Laurent: Evolutionary Conditions for the Emergence of Communication in Robots. In: *Current Biology* 17 (2007), Nr. 1-6
- [HTP<sup>+</sup>05] HOEN, Pieter J. ; TUYLS, Karl ; PANAIT, Liviu ; LUKE, Sean ; POUTRÉ, Johannes A. L.: An Overview of Cooperative and Competitive Multiagent Learning. In: *LAMAS*, 2005 (Lecture Notes in Computer Science), S. 1–46
- [Kar06] KARWOWSKI, Waldemar: *International Encyclopedia of Ergonomics and Human Factors*. Boca Raton : The CRC Press, 2006

- [Kie96] KIENPOINTNER, Manfred: Whorf and Wittgenstein. Language, world view and argumentation. In: *Argumentation* (1996), Nr. 4, S. 475–494
- [KJKL08] KÖNIG, Lukas ; JEBENS, Kristof ; KERNBACH, Serge ; LEVI, Paul: Stability of On-Line and On-Board Evolving of Adaptive Collective Behavior. In: *Proc. of Second IEEE International Conference on Self-Adaptive and Self-Organizing Systems* (2008), S. 293–302
- [KS08] KÖNIG, Lukas ; SCHMECK, Hartmut: Evolving Collision Avoidance on Autonomous Robots. In: *Biologically-Inspired Collaborative Computing*. Boston : Springer, 2008 (IFIP International Federation for Information Processing), S. 85–94
- [Lam09] LAMARCK, Jean Baptiste Pierre A. d.: *Philosophie zoologique, ou, Exposition des considerations relative a l’histoire naturelle des animaux*. Paris, 1809
- [Len03] LENZEN, Manuela: *Evolutionstheorien in den Natur- und Sozialwissenschaften*. Frankfurt/Main : Campus Verlag, 2003
- [Lip07] LIPSON, Hod: Evolutionary Robotics: Emergence of Communication. In: *Current Biology* 17 (2007), Nr. 9
- [LK58] LINDAUER, Martin ; KERR, Warwick E.: Die gegenseitige Verständigung bei den stachellosen Bienen. In: *Journal of Comparative Physiology A: Neuroethology, Sensory, Neural, and Behavioral Physiology* (1958), Nr. 41, S. 405–434
- [Moo56] MOORE, E. F.: Gedanken experiments on sequential machines. In: SHANNON, C. E. (Hrsg.) ; MCCARTHY, J. (Hrsg.): *Automata Studies*. Princeton U., 1956, S. 129–153
- [NF00] NOLFI, Stefano ; FLOREANO, Dario: *Evolutionary robotics : the biology, intelligence, and technology of self-organizing machines*. Cambridge : MIT Press, 2000
- [Sch06] SCHÖNENBERGER, Helmut: *Kommunikation und Unternehmertum*. Wiesbaden : Deutsche Universitäts-Verlag, 2006
- [Ter98] TERHARDT, Ernst: *Akustische Kommunikation, Grundlagen mit Hörbeispielen*. Berlin Heidelberg : Springer, 1998

- [Tri08] TRIANNI, Vito: *Studies in Computational Intelligence*. Bd. 108: *Evolutionary Swarm Robotics*. Berlin Heidelberg : Springer, 2008
- [Wea49] WEAVER, Warren: Recent Contributions to the Mathematical Theory of Communication. In: *The mathematical theory of communication*. University of Illinois Press, 1949, S. 93–117
- [Wei07] WEICKER, Karsten: *Evolutionäre Algorithmen*. Leipzig : B.G. Teubner Verlag / GWV Fachverlage GmbH, 2007
- [WGW03] WALKER, Joanne ; GARRETT, Simon ; WILSON, Myra: Evolving controllers for real robots: A survey of the literature. In: *Adaptive Behavior* 11 (2003), Nr. 3, S. 179–203
- [WK10] WERTH, Carolina ; KERSTHOLT, Fabian: *Organic Computing Learning Robots Seminar WS 2009/2010*. 2010. – Evolving Communication on Autonomous Robots
- [WKF09] WAIBEL, Markus ; KELLER, Laurent ; FLOREANO, Dario: Genetic team composition and level of selection in the evolution of cooperation. In: *IEEE Transactions on Evolutionary Computation* 13 (2009), Nr. 3, S. 648–660





## Erklärung

Ich versichere hiermit wahrheitsgemäß, die Arbeit bis auf die dem Aufgabesteller bereits bekannte Hilfe selbständig angefertigt, alle benutzten Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten anderer unverändert oder mit Abänderungen entnommen wurde.

Karlsruhe, den 28.07.2010,

Fabian Kerstholt